VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Aplikace pro monitorování Android zařízení

Application for Android Device Monitoring

2019 Bc. Erika Čížová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Erika Čížová**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 1801T064 Informační a komunikační bezpečnost

Téma: Aplikace pro monitorování Android zařízení
Application for Android Device Monitoring

Jazyk vypracování: čeština

Zásady pro vypracování:

Práce se zabývá bezpečností a monitoringem operačního systému Android. Cílem bude vytvořit řešení, které může být užito jako dohledová aplikace pro mobilním zařízením. Tato aplikace by měla být vhodná např. pro dohled rodičů nad Android zařízeními svých dětí. Řešení se bude skládat ze dvou softwarových částí. První bude klientská aplikace, která bude spuštěna na zařízení dítěte. Druhou bude serverové řešení umožňující rodičům vzdáleně spravovat a monitorovat spárovaná zařízení.

Hlavní body zadání:
1. Seznámení se s problematikou vývoje aplikací pro operační systém android. Student by se měl zaměřit na funkce, které by mohly být použity pro zvýšení bezpečnosti uživatele aplikace.
2. Rešerše již existujících řešení.
3. Implementace monitorovací aplikace pro operační systém Android a implementace serverové části. Serverová část bude sloužit k monitoringu a k provádění změn v nastavení mobilní aplikace.
4. Testování řešení a srovnání s existujícími nástroji.

Seznam doporučené odborné literatury:

[1] R. Meier, Professional Android 4th Edition, Wrox, 2018, ISBN: 978-1118949528
[2] M. Yener, O. Dunar, Expert Android Studio, Wrox, 2016, ISBN: 978-1119089254

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Plucar, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Čestné prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě dne 26. 4. 2019 .....

podpis

Poděkování

Na tomto místě bych ráda poděkovala panu Ing. Janovi Plucarovi, Ph.D. za odborné vedení práce, cenné rady a vstřícnost při konzultacích a vypracovávání této diplomové práce.

Abstrakt

Práce se zabývá bezpečností a monitoringem operačního systému Android. Cílem je vytvořit řešení, které může být využito jako dohledová aplikace mobilního zařízení. Tato aplikace by měla umět posoudit bezpečnost instalovaných aplikací na zařízení a mohla by tak být vhodná např. pro dohled rodičů nad Android zařízeními svých dětí.

Řešení se skládá ze dvou softwarových částí. První je klientská aplikace, která bude spuštěna na monitorovaném zařízení, tedy například na zařízení dítěte. Druhou částí je serverové řešení umožňující ohodnocení bezpečnosti aplikací v souvislosti na udělených oprávnění aplikací. Rodičům bude umožňovat vzdáleně monitorovat spárovaná zařízení.

Klientská část aplikace je naprogramována v programovacím jazyce Kotlin, serverová část aplikace pak v programovacím jazyce Java. První část této práce je věnována popisu existujících aplikací zabývajících se touto nebo podobnou problematikou, následně pak bezpečností a popisu technologií monitoringu zařízení s operačním systémem Android. Poslední část práce je věnována popisu návrhu, implementace a testování vzniklé aplikace.

Klíčová slova

Android, Java, Kotlin, root zařízení, bezpečnost zařízení s operačním systémem Android, monitorování zařízení, nebezpečí skrývané v udělovaných oprávněních.

Abstract

This thesis deals with the security and monitoring of the device with Android operating system. The goal is to create a solution that can be used as a mobile application surveillance application. This application should be able to assess the security of installed applications on the device and thus could be useful for example for parental supervision over children's Android devices.

The solution consists of two software parts. The first is a client application that will run on a monitored device, such as a child's device. The second part is a server solution that allows security evaluation of installed applications in context of granted permissions. Parents should then be able to remotely monitor paired devices.

The client part of the application is programmed in the Kotlin programming language, the server part of the application is programmed in the Java programming language. The first part of this work is dedicated to the description of existing applications that are dealing with this issue, and then to the security and description of the monitoring technologies of devices with Android operating system. Last part of this work is dedicated to the design, implementation and testing of created application.

Key Words

Android, Java, Kotlin, device rooting, Android device security, device monitoring, granting permissions and their security context.

Obsah

Seznam použitých symbolů a zkratek

OS	Operating system
SDK	Software Development Kit
GUI	Graphical User Interface
UI	User Interface
SQL	Structured Query Language
XML	eXtensible Markup Language
UML	Unified Modeling Language
AES	Advanced Encryption Standard
RSA	iniciály autorů Rivest, Shamir, Adleman
DSA	Digital Signature Algorithm
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
HTTPS	Hypertext Transfer Protocol Secure
OEM	Original Equipment Manufacturer
DRM	Digital Rights Management
NFC	Near-field communication
OHA	Open Handset Alliance
GPL	General Public License
ART	Android Runtime
AOT	Ahead-of-time
JIT	Just-in-time
DEX	Dalvik Executable
APK	Android Package File
URL	Uniform Resource Locator
REST	Representational State Transfer
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
ADB	Android Debug Bridge
JPA	Java Persistence API
ORM	Object-Relational Mapping

POJO	Plain Old Java Object
MIT	Massachusetts Institute of Technology

Seznam ilustrací

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Mobilní zařízení se v dnešní době stalo součástí života snad každého z nás. Díky mobilním technologiím máme možnost být v neustálém spojení se zbytkem světa, s rodinou či přáteli. Jsou ale naše data zadávána do mobilních zařízení v bezpečí? Neudělují uživatelé mobilních zařízení mnohdy nevědomky práva aplikacím využívat data, která pro svůj provoz nepotřebují?

OS (Operating System) Android je mezi mobilními operačními systémy již 10 let a aktuální verze je Android 9.0 s označením Pie. Samotná platforma Android však dává k dispozici nejen operační systém s uživatelským prostředím pro koncové uživatele, ale i kompletní řešení nasazení operačního systému pro mobilní operátory a výrobce zařízení. V neposlední řadě poskytuje pro vývojáře aplikací efektivní nástroje pro jejich vývoj (SDK - Software Development Kit).

Obsahem této diplomové práce je mapování problematiky bezpečnosti OS Android a popis technologií monitoringu zařízení s operačním systémem Android. Celá práce je zaměřena především na problematiku udělovaných přístupových práv (oprávnění) aplikacím. Tomuto následuje popis řešení implementace aplikace s názvem Permission Control pro monitorování zařízení s OS Android. Kapitoly věnované vyvíjené aplikaci obsahují popis tří základních komponent aplikace. Jedná se v první řadě o popis databáze, která slouží pro uchovávání informací o testovaných aplikacích. Dále je uveden popis serverové části aplikace, která slouží ke komunikaci s databází a k výpočtu skóre aplikace spolu s její klasifikací. Jako poslední je uveden popis návrhu i implementace klientské části výsledné aplikace, sloužící především pro zobrazování výsledků testování vybrané aplikace. Aplikace obsahuje možnost zobrazení otestovaných aplikací jiného uživatele, čímž nastiňuje problematiku spojenou s dohledem rodičů nad zařízeními svých dětí.

Následující kapitola nastiňuje aktuální stav problematiky monitorování chování uživatele na OS Android, včetně udělování přístupových práv aplikacím. Třetí kapitola je pak zaměřena na architekturu OS Android, ve které jsou popsány základní součásti OS, jako je jádro, aplikační runtime a knihovny. O bezpečnosti OS Android pojednává čtvrtá kapitola, kde jsou vysvětleny hlavní součásti bezpečnostního modelu OS Android spolu s vysvětlením problematiky týkající se tzv. rootování zařízení. Pátá kapitola je věnována přehledu existujících oprávnění v OS Android, a to včetně popisu skupin oprávnění a jejich závažnosti. Šestá kapitola je zaměřena na popis komponent, které jsou v aplikacích využívány. K praktické části diplomové práce se dostáváme spolu s kapitolami sedm až devět, kde dochází k popisu návrhu aplikace, a to obou jejich částí včetně návrhu využívané databáze. Následně, v osmé kapitole, která je věnována popisu implementace, jsou rozebrány klíčové prvky obou částí aplikace. V deváté kapitole, která je zároveň poslední kapitolou v praktické části práce, je uveden postup testování výsledné aplikace, které se zaměřuje především na procento správně ohodnocených testovaných aplikací.

2 Seznámení s problematikou

Mobilní aplikace požadují nejrůznější přístupová oprávnění k prostředkům zařízení. Tato skutečnost může být využita vývojáři aplikací k tomu, aby ohrozili soukromí uživatelů pro své vlastní zisky. Data uživatelů pak mohou být poskytnuta třetím stranám, a to aniž by o tom uživatel věděl, nebo k tomu dal vědomý souhlas. Tyto aplikace pak mohou být využity pro sledování chování daného uživatele a jeho činností, a to nejen v rámci internetu, ale také například jeho lokace v reálném prostředí. Aplikace totiž požádá uživatele o oprávnění, která nemusí být nutně spjata s krádeží dat nebo s odhalováním soukromí uživatele. Často se lze setkat s tím, že získání citlivých informací od uživatelů je manipulováno společnostmi prostřednictvím aplikací a oprávnění, která jsou jimi požadována. Vzhledem k pomalému přizpůsobování uživatelů k neustále se měnícím technologiím je zapotřebí nástroje, který bude posuzovat rizika oprávnění, která jsou aplikacím udělována.

2.1 Pochopení rizik udělovaných oprávnění

Následující tabulka (viz Tabulka 1: Nebezpečná oprávnění [1][2]) zobrazuje riziková a nebezpečná oprávnění udělována aplikacím dle zdrojů [1][2]. Použití těchto oprávnění v aplikaci samozřejmě nemusí hned znamenat, že aplikace zasílá nebo jinak zneužívá data, které jí uživatel poskytuje. Podle zdrojů [1] a [2] se však jedná o oprávnění nebezpečná a zneužitelná pro škodlivou činnost. Oprávnění udělovaná aplikacím se dělí do skupin, která souvisí s funkcemi a možnostmi zařízení (například skupina `CALENDAR` obsahuje oprávnění týkající se čtení a zápisu dat z nebo do kalendáře).

Tyto skupiny oprávnění jsou následně řazeny do několika úrovní ochrany, které ovlivňují, zda jsou vyžadovány požadavky na oprávnění za běhu aplikace. Existují tři úrovně ochrany, které ovlivňují aplikace třetích stran: normální, podpisové a nebezpečné oprávnění (více k tomuto tématu v 5. kapitole).

Operační systém Android obsahuje vyspělé mechanismy zabezpečení na základě svého modelu oprávnění a bezpečnostních funkcích zděděných z jádra Linuxu (více v kapitole 3). Díky těmto mechanismům mají aplikace třetích stran omezený přístup ke zdrojům zařízení. Za normálních okolností mohou aplikace přistupovat k chráněným prostředkům pouze v případě, že mají udělená oprávnění.

Naneštěstí OS Android neobsahuje žádný vestavěný ověřovací systém, který by kontroloval, zda aplikace nepožadují příliš mnoho (nebo málo) oprávnění, což by mohlo vést k vážným problémům týkajících se kvality aplikace a/nebo ochrany soukromí. Aplikace požadující příliš mnoho oprávnění pak mohou vytvářet zbytečné zranitelnosti, které zanechávají potenciál pro zneužití SDK v rámci aplikace nebo v rámci jiných aplikací nainstalovaných v zařízení.

Tabulka 1: Nebezpečná oprávnění [1][2]

Skupiny oprávnění	Jednotlivá oprávnění	Popis
CALENDAR	READ_CALENDAR WRITE_CALENDAR	Správa kalendáře
CALL_LOG	READ_CALL_LOG WRITE_CALL_LOG PROCESS_OUTGOING_CALLS	Správa telefonních hovorů a záznamů o nich
CAMERA	CAMERA	Pořizování fotografií a nahrávání videa
CONTACTS	READ_CONTACTS WRITE_CONTACTS GET_ACCOUNT	Správa kontaktů
LOCATION	ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION	Zjištění aktuální polohy zařízení
MICROPHONE	RECORD_AUDIO	Záznam audia
PHONE	READ_PHONE_STATE READ_PHONE_NUMBERS CALL_PHONE ANSWER_PHONE_CALLS ADD_VOICEMAIL USE_SIP	Vytáčení a správa telefonních hovorů
SENSORS	BODY_SENSORS	Srdeční frekvence apod.
SMS	SEND_SMS RECEIVE_SMS READ_SMS RECEIVE_MMS RECEIVE_WAP_PUSH	Posílání a čtení zpráv
STORAGE	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE	Přístup k fotografiím, médiím a souborům

Spolu s příchodem verze Androidu 6.0 s označením Marshmallow došlo k důležité a podstatné změně struktury oprávnění. V předchozích verzích bylo nutné přijetí všech oprávnění dané aplikace k tomu, aby byla dokončena instalace. Od verze 6.0 dál má však uživatel možnost přijmout jen některé, pro běh aplikace důležité, oprávnění a další může udělit až za běhu aplikace. Uživateli je tak dána větší kontrola nad udělovanými oprávněními aplikací.

Existuje zde však jeden obrovský a známý problém – běžní uživatelé mobilních aplikací si sice přečtou obrazovky s informacemi o oprávněních, které daná aplikace vyžaduje, většinou však nerozumí důsledkům uděleného souhlasu. Podle výzkumu [3] nejsou uživatelé připraveni na taková rozhodnutí ani při instalaci, ale dokonce ani při prvním spuštění aplikace.

2.2 Související práce

Následující tabulka (viz Tabulka 2: Existující nástroje nebo aplikace zabývající se analýzou oprávnění aplikací) zobrazuje existující nástroje pro analýzu udělovaných oprávnění aplikacím [2].

Tabulka 2: Existující nástroje nebo aplikace zabývající se analýzou oprávnění aplikací

Název nástroje / aplikace	**Prostředí**	**Funkce**
M-Perm [4]	Virtuální stroj	Volání API
DroidScope [5]	Emulátor	Volání API, systémová volání, instrukce Dalviku
DroidBox [6]	Emulátor	Volání API

Z tabulky vyplývá, že tyto zmíněné nástroje nejčastěji využívají pro analýzu udělovaných oprávnění volání API (Application Programming Interface), případně systémová volání. Tímto dochází k odhalení základního chování sledovaných aplikací. Všechny tyto existující nástroje jsou navíc založeny na prostředí běžícím tzv. v laboratoři, neboli na virtuálním stroji nebo emulátoru, což může způsobit nepřesnosti analýzy, jelikož aplikace nemusí v emulovaném prostředí fungovat stejně, jako na reálném zařízení.

V rámci mého výzkumu existujících aplikací a nástrojů jsem narazila na velké množství řešení, která požadovaly zařízení v tzv. root módu. Rozhodla jsem se proto k nasměrování této práce k takovému řešení, které nebude vyžadovat root mód a navíc bude celé řešení implementováno jako mobilní aplikace, která bude kontrolovat práva ostatních aplikací a upozorňovat uživatele na možné bezpečnostní rizika.

Pro detailní pokrytí existujících aplikací na mobilní zařízení s podobným zaměřením jsem provedla průzkum obchodu Google Play. Došla jsem ke zjištění, že již existuje mnoho aplikací zabývajících se správou oprávnění, avšak žádná z nalezených aplikací neposkytuje možnost porovnání udělených oprávnění v usvědčených škodlivých aplikacích s udělenými oprávněními testovaných aplikací. Nalezené aplikace zabývající se touto problematikou neposkytují výpočet skóre aplikace, které je založeno na udělených oprávněních. Z tohoto důvodu jsem se rozhodla směrovat tuto práci a výslednou aplikaci tímto nastíněným směrem.

3 Operační systém Android

Základ platformy Android byl vyvinut společností Android Inc., kterou v roce 2005 odkoupila firma Google Inc. V roce 2007 pak byla založena Open Handset Alliance (OHA), která původně sdružovala několik výrobců hardware, software a telekomunikační společnosti podílejících se tímto na prosazování otevřených standardů ve světe mobilních zařízení. Od počátku roku 2008, kdy byla vydána první veřejně dostupná verze platformy Android, jsou všechny její součásti k dispozici pod licencí Apache a GPL v2 (General Public Licence). Jedná se tak o tzv. open source software.

Operační systém Android je aktuálně nejprodávanějším mobilním operačním systémem. Ve druhém kvartálu roku 2017 dosáhl 87,8% podílu na trhu [7]. S takovou popularitou však přicházejí nejrůznější hrozby.

Uživatelé tento operační systém vyhledávají nejen pro jeho úplnou propojenost s Google účtem, ale také pro lepší cenovou dostupnost nebo pro velkou nabídku nejrůznějších aplikací, které jsou z velké části dostupné zdarma. Dalším důvodem může být vcelku jednoduchá tvorba nových aplikací v případě, že je uživatel zaměřen tímto směrem.

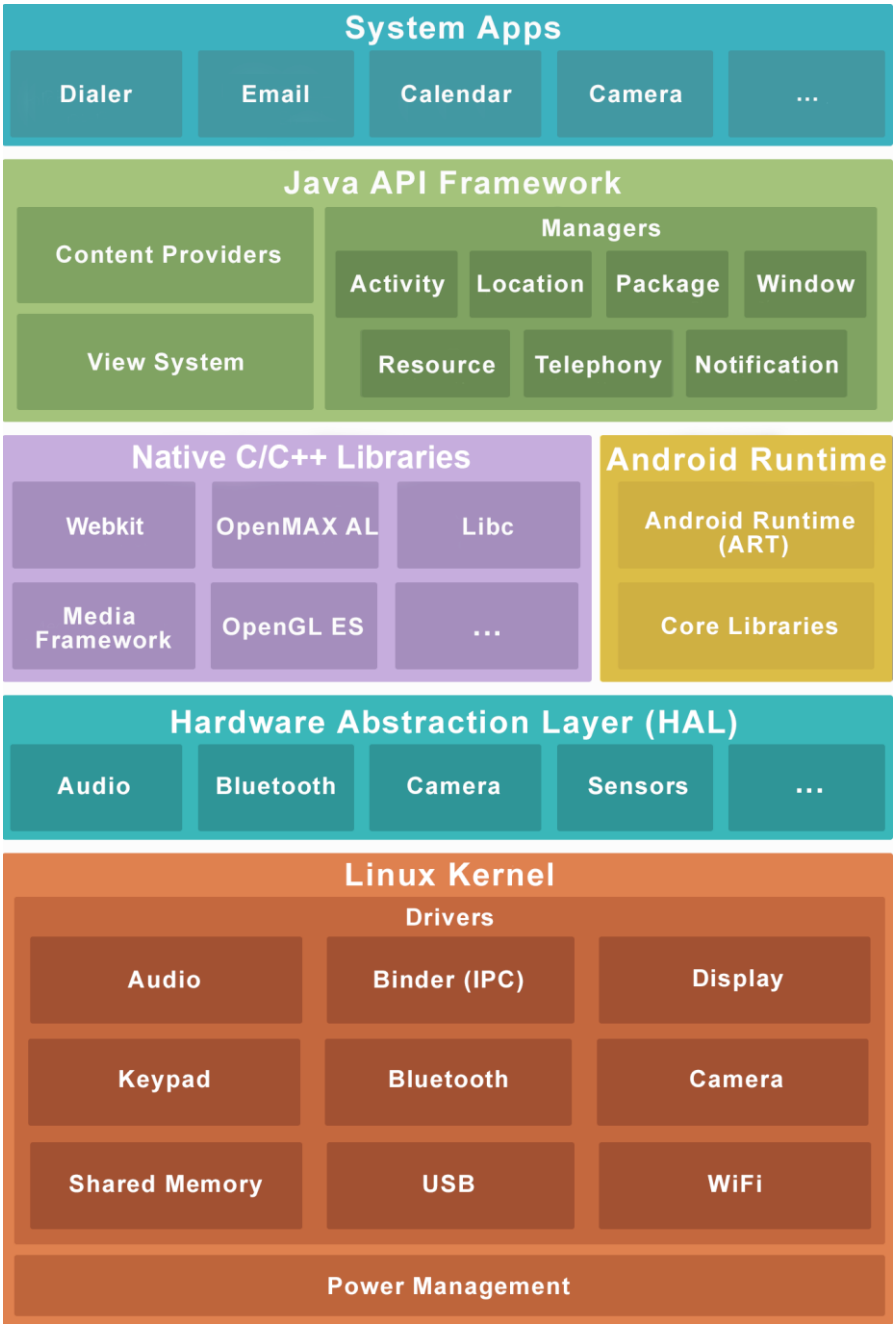
3.1 Architektura OS Android

Operační systém Android je tzv. open source platforma vytvořena pro širokou škálu mobilních zařízení, která je založena na linuxovém jádru. Využívá vyspělý a hlavně různorodý hardware a software a vlastní i sdílená data. Je proto nutné, aby měl systém k dispozici takové prostředí, které zajistí zabezpečení uživatelů, jejich dat, aplikací a využívané sítě. [8]

Při vývoji OS Android byl kladen důraz na jednoduchost tvorby aplikací. Vývoj aplikací pro OS Android je pro vývojáře usnadněn flexibilitou bezpečnostních prvků. Pokud vývojář není dostatečně informován v otázce bezpečnosti OS Android, je výchozí nastavení bezpečnosti dostatečně elastické, aby jeho neznalost nebyla překážkou.

Aby byla skutečně zajištěna bezpečnost otevřené platformy, je potřeba rozsáhlá bezpečnostní architektura a do detailu vyladěné bezpečnostní zásady. Proto OS Android disponuje vícevrstvou ochranou, což zajišťuje nejen pružnost platformy, ale i dostatečnou ochranu jejich uživatelů.

Architektura operačního systému Android je rozdělena do 6 vrstev, kdy každá vrstva plní svůj účel. Není ovšem vždy nutné, aby tyto vrstvy byly přímo odděleny jedna od druhé. Obrázek 1 zobrazuje uspořádání jednotlivých součástí platformy Android, které jsou dále rozebírány.



Obrázek 1: Architektura OS Android [8]

3.1.1 Jádro operačního systému

Nejnižší vrstvou architektury je jádro operačního systému a tvoří tak abstraktní vrstvu mezi používaným hardwarem a zbytkem softwaru ve vyšších vrstvách. Jádro Androidu je postaveno na upraveném Linuxu, a to na verzi 2.6.x. Pozdější verze Androidu už však využívají novější linuxová jádra.

Android využívá mnoha vlastností Linuxu, jako je podpora správy paměti, správy sítí, zabudované ovladače nebo správy procesů pro umožnění souběžného běhu několika aplikací. Tyto aplikace pak běží jako samostatné procesy s oprávněními stanovenými systémem. Tyto

vlastnosti Linuxu přispívají ke stabilitě a ochraně systému. Důvodem pro využití jádra Linux byla vlastnost poměrně snadného sestavení na různých zařízeních a tím zaručená přenositelnost. Systém tak může být použit bez ohledu na použitý hardware, čipovou sadu, velikost nebo rozlišení obrazovky.

Klíčové rysy bezpečnosti převzaté z linuxového jádra pak obsahují model oprávnění založený na aktuálním uživateli, izolaci procesů pomocí aplikačního sandboxu, rozšiřitelný mechanismus pro bezpečnou mezi procesorovou komunikaci, schopnost odstranit zbytečné a potenciálně nebezpečné části jádra. [8]

3.1.2 Hardware Abstraction Layer (HAL)

Hardwarová abstraktní vrstva (HAL) definuje a poskytuje standardní rozhraní, které odhaluje možnosti hardwaru daného zařízení na vyšší úrovně Java API rámce. Modul HAL se skládá z několika knihoven, z nichž každá implementuje rozhraní pro konkrétní typ hardwarové komponenty (jako je kamera nebo modul bluetooth). V případě, že rozhraní Java API framework vyvolá přístup k hardwaru zařízení, systém Android načte knihovní modul pro tuto hardwarovou komponentu. Použití HAL tedy umožňuje implementovat funkce hardwaru bez ovlivnění nebo úpravy systému ve vyšší úrovni. [8]

3.1.3 Nativní C/C++ knihovny

Další vrstvou architektury OS Android jsou knihovny, které jsou napsány v C nebo C++ kódu a jsou využívány nejrůznějšími komponentami systému. Funkce z těchto knihoven jsou dostupné vývojářům prostřednictvím Android API Framework (více o tématu v kapitole 2.1.5).

Mezi základní knihovny pak patří knihovny pro podporu přehrávání video a audio formátů (tzv. Media Libraries), následně pak knihovna webového prohlížeče, který podporuje i vložení náhledu webových stránek (LibWebCore), knihovna pro relační databáze (SQLite) a mnoho dalších (OpenSSL, OpenGL ES, Libc, atp.). [8]

3.1.4 Android runtime

Aplikace na OS Android jsou nejčastěji psané v programovacím jazyce Java, ačkoliv v poslední době se můžeme stále častěji setkat i s programovacím jazykem Kotlin. Hlavní odlišností ve vývoji aplikací pro mobilní zařízení oproti vývoji pro desktopové zařízení je stavba architektury a omezení paměti a procesoru zařízení, na kterých je program pro Android spouštěn.

OS Android do verze 4.4 obsahoval virtuální stroj Dalvik, na kterém jsou spouštěny všechny aplikace OS Android, a tvoří tak abstraktní vrstvu mezi operačním systémem a počítačem a úzce souvisí s virtuálním strojem Javy.

Od verze 4.4 došlo k náhradě tohoto virtuálního stroje za Android Runtime (ART). Hlavními přínosy této změny byla kompilace tzv. Ahead-of-time (AOT), tedy dopředná kompilace, nebo tzv. Just-in-time (JIT), tedy kompilace až za běhu aplikace. Mezi další hlavní rysy ART patří optimalizace tzv. garbage collectoru nebo vylepšení vývoje a ladění aplikací. Touto změnou došlo ke značné úspoře energie a k výraznému zrychlení aplikací. ART je totiž stylizován pro spouštění

více virtuálních strojů na zařízení s malou pamětí, a to za spouštění souborů DEX (Dalvik Executable). DEX je bajtkódový formát navržený speciálně pro Android, který je optimalizován pro minimální nároky na paměť zařízení. [8]

3.1.5 Java API framework

Vrstva API framework je nejdůležitější pro vývojáře. Poskytuje přístup k velkému počtu služeb a funkcí OS Android, které mohou být využity přímo v aplikacích. Tyto služby mohou zpřístupňovat data v jiných aplikacích, prvky uživatelského rozhraní, upozorňovací stavový řádek, aplikace běžící na pozadí, hardware používaného zařízení a mnoho dalších funkcí.

Základní sada služeb zahrnuje především:

- Bohatou sadu prvků View, který je možné použít pro sestavení uživatelského rozhraní aplikace, a to včetně seznamů, textových polí, tlačítek, zaškrtávacích tlačítek a dokonce i vestavěný webový prohlížeč.
- Content providers, neboli poskytovatelé obsahu, umožňující aplikacím přístup k datům (např. kontakty) jiných aplikací.
- Resource manager, který poskytuje přístup „nekódovým“ zdrojům, jako jsou lokalizované řetězce, grafika nebo soubory s rozložením komponent.
- Notification manager, neboli manažer upozornění, který umožňuje všem aplikacím zobrazit vlastní upozornění ve stavovém řádku.
- Activity manager, neboli správce aktivit, řídící životní cyklus aplikací a poskytující orientaci v zásobníku s aplikacemi.

Toto rozhraní API tvoří základní stavební blok, který je zapotřebí k vytváření aplikací na OS Android. Vývojáři mají plný přístup ke stejnému API, které je využíváno i systémovými aplikacemi. [8]

3.1.6 Základní aplikace

Nejvyšší vrstvu systému tvoří aplikace, které využívají běžní uživatelé. Může jít o aplikace předinstalované nebo dodatečně stažené z Google Play. Aplikace zahrnuté do platformy nemají žádný speciální příznak nebo status a je tedy možné je téměř všechny bez problémů nahradit aplikacemi třetích stran (nebo i naší vlastní).

Základní systémové aplikace pak fungují dvěma způsoby. Jedním je klasické využití uživatelem, zároveň však poskytují klíčové vlastnosti, které mohou využít developeři pro vlastní aplikace. Například pokud máme vlastní aplikaci, která bude doručovat zprávy SMS, nemusí developer znovu vyvíjet funkce pro posílání zpráv. Namísto toho stačí zavolat kteroukoliv aplikaci, která je již nainstalována v zařízení a pomocí ní tuto SMS zprávu poslat. [8]

3.2 Správa napájení

Operační systém Android má zabudovanou vlastní správu napájení (nad standardní správou napájení, kterou obsahuje jeho Linuxové jádro). Tato správa napájení je navržena s předpokladem, že procesor by neměl spotřebovávat energii v případě, že žádné aplikace nebo služby nevyžadují napájení.

Android požaduje, aby aplikace a služby využívaly funkci „zákazu úsporného režimu“ (z angličtiny „wake lock“) pro vyžádání zdrojů procesoru, a to prostřednictvím nativních knihoven Linuxu nebo Android API rámce (z angličtiny „framework“). V případě, že v systému nejsou zavedeny žádné aktivní zákazy úsporného režimu, Android je nastaven tak, aby došlo k vypnutí procesoru. [9]

Operační systém Android obsahuje následující vylepšení týkající se výdrže baterie: [10]

- Omezení aplikací - existují aplikace, které mají negativní vliv na životnost baterie, uživateli je tak poskytnuta možnost omezení aplikace na spotřebu zdrojů procesoru. Ve výchozím nastavení nemají aplikace žádné omezení pro běh na pozadí.
- Pohotovostní režim aplikace - aktuálně nevyužívané aplikace je možno umístit do tzv. pohotovostního režimu. Takovýmto aplikacím je dočasně omezen přístup k síti, čímž je odložena synchronizace a další úlohy.
- Doze režim - v případě, že uživatelé nepoužívají aktivně své zařízení po delší dobu (obrazovka je vypnutá a stacionární) může celý operační systém Android vstoupit do stavu hlubokého spánku (s pravidelným obnovením běžných operací).
- Výjimky - Předinstalované systémové aplikace a služby cloudových zpráv jsou zpravidla ve výchozím nastavení vyňaty z pohotovostního a doze režimu. Vývojáři aplikací mohou používat tzv. intenty k uplatnění stejných nastavení ve svých aplikacích. Uživatelé pak mohou v nabídce Nastavení vyloučit aplikace z režimů Úspora pohotovostního režimu a Úspora energie.

3.2.1 Zákaz úsporného režimu

Aby nedocházelo k rychlému vyčerpání baterie daného zařízení, obsahuje operační systém Android možnost snížení spotřeby energie. Dochází k tomu tím, že při delší nečinnosti přejde zařízení do stavu spánku. Existují však chvíle, kdy aplikace potřebuje zařízení probudit (rozsvítit obrazovku, využít procesor) a udržet ho tak v bdělém stavu pro dokončení nějaké práce.

Funkce zákazu vstoupení do úsporného režimu je využívána pro přístup ke zdrojům procesoru. V závislosti na typu zákazu je pak systému odepřena možnost vstoupení do režimu úspory napájení. [9]

Aktuálně jsou v systému Android využívány následující dva typy zákazu úsporného režimu: `WAKE_LOCK_SUSPEND`, který brání celému systému v uspání a `WAKE_LOCK_IDLE`, který brání systému v přejití do stavu nízké spotřeby energie.

4 Bezpečnost OS Android

Jednou z velkých výhod zařízení s OS Android je možnost disponovat velkým množstvím aplikací a podporovat službu cloudu. To je zásluhou rozšířeného bezpečnostního modelu, Android obsahuje nejrůznější bezpečnostní funkce a spolupracuje s mnoha vývojáři na udržení bezpečnosti platformy a jejího ekosystému. Robustní bezpečností model je nezbytný k tomu, aby umožnil energetický ekosystém aplikací a zařízení postavených na platformě Android. Výsledkem je, že je systém Android po celý svůj vývojový životní cyklus podroben přísnému bezpečnostnímu programu obsahující nejrůznější testy.

Inspirací k tomu, na co se v těchto testech zaměřit, se jim staly chyby ostatních mobilních, desktopových a serverových platforem. Vytváří tak silný zabezpečovací program, jehož úkolem je pohotově reagovat na bezpečnostní problémy a předcházet slabým místům, která byla vypozorována u konkurence.

Android je od základu navržen tak, aby byl otevřený, tzv. open-source. Aplikace pro Android využívají pokročilý hardware a software, lokální i serverová data, které je zapotřebí chránit. Z tohoto důvodu platforma nabízí aplikační prostředí, které chrání důvěrnost, integritu a dostupnost uživatelů, dat, aplikací, zařízení a sítě.

Zabezpečení otevřené platformy vyžaduje silnou bezpečnostní architekturu (popsanou v kapitole 2.1) a přísné bezpečnostní programy. Android byl navržen s vícestránkovým zabezpečením, které je dostatečně flexibilní, aby podporovalo otevřenou platformu a současně ochránilo všechny uživatele platformy.

Android je určen pro vývojáře. Bezpečnostní kontroly byly navrženy tak, aby snížily zátěž pro vývojáře. Vývojáři, kteří umí vyvíjet bezpečné aplikace, mohou snadno pracovat s flexibilními bezpečnostními prvky, které jsou součástí systému, a spoléhat se na ně. Vývojáři méně obeznámeni s bezpečností jsou chráněni bezpečnými výchozími nastaveními.

Vedle poskytnutí stabilní platformy Android poskytuje vývojářům další podporu mnoha způsoby. Tým, zabývající se bezpečností ve společnosti Android, hledá potenciální zranitelnosti v aplikacích a navrhuje způsoby, jak tyto problémy odstranit. Android proto poskytuje aktualizace zabezpečení pro kritické knihovny softwaru (jako je například OpenSSL, které slouží k zabezpečení komunikace aplikací).

Android je určen pro uživatele. Těm je zajištěna viditelnost oprávnění požadovaných každou z jejich aplikací a možná kontrola těchto oprávnění. Tento bezpečností prvek zahrnuje očekávání, že se útočníci pokusí provést běžné útoky, jako jsou útoky sociálního inženýrství, aby přesvědčili uživatele zařízení k instalaci malwaru. Android byl však navržen tak, aby snížil pravděpodobnost těchto útoků a výrazně omezil dopad útoku v případě, že byl úspěšný. [11]

4.1 Klíčové vlastnosti bezpečnostního modelu OS Android

Kontrola návrhu (designu)

Již v rané fázi vývoje životního cyklu platformy Android byl vytvořen proces zabezpečení, a to díky bohatého a konfigurovatelného bezpečnostního modelu a designu. Každá nová funkce platformy je přezkoumána inženýrskými bezpečnostními prostředky. Pouze ty funkce platformy, které projdou všemi testy, jsou integrovány do architektury systému. [11]

Penetrační testování a kontrola kódu

Penetrační testy jsou simulace pokusů o útok na aplikaci zvenku i zevnitř pomocí různých nástrojů. Komponenty, které byly vytvořeny jako součást vývoje otevřené platformy Android, podléhají bezpečnostním hodnocením. Tato hodnocení provádí bezpečnostní tým Androidu, tzv. Android Security Team, a následně nezávislí bezpečnostní konzultanti. Cílem těchto hodnocení je identifikovat slabá místa bezpečnostního modelu a možné zranitelnosti systému či aplikace, a to s dostatečným předstihem. Následně je cílem simulovat typy analýz, které budou prováděny externími bezpečnostními experty po uvolnění platformy. [11]

Open Source a Společenství

Všechny zúčastněné strany projektu Android Open Source (firma Google, vývojáři, bezpečnostní technici, uživatelé, aj.) mohou využívat rozsáhlé bezpečnostní kontroly. Mezi možnosti kontroly patří i fórum Google Play, kde firmy a vývojáři můžou poskytovat informace o konkrétních aplikacích. [11]

Reakce na události

I přes všechna bezpečnostní opatření vývojářů OS Android mohou nastat potíže, a proto Android vytvořil souhrnné bezpečnostní odezvy, které řeší vzniklé problémy. Tým starající se o bezpečnost OS Android neustále monitoruje specializované i veřejně známé bezpečnostní problémy pomocí diskuzí (týkajících se možných chyb v zabezpečení), nacházejících se na různých diskuzních portálech. Po objevení legitimních problémů má tým Android reakční proces, který umožňuje rychlé zmírnění zranitelných míst, aby se zajistilo, že potenciální riziko bude minimalizováno pro všechny uživatele systému Android. Tyto reakce jsou prováděny přes cloud pomocí aktualizace platformy OS Android, odstraněním aplikace z Google Play nebo odstraněním aplikace přímo ze zařízení, ve kterém je nainstalována. [11]

4.2 Bezpečnostní služby společnosti Google

Společnost Google poskytuje sadu služeb založených na cloudových službách, které jsou k dispozici pro kompatibilní zařízení Android (zařízení se službami Google Mobile Services). Zatímco tyto služby nejsou součástí projektu Android Open Source, jsou součástí mnoha zařízení Android. [11]

Mezi primární bezpečnostní služby společnosti Google se řadí:

- Služba **Google Play**, která umožňuje uživatelům objevovat, instalovat a nakupovat nové aplikace. Poskytuje také možnost napsání recenze, ověření licencí aplikací a skenování zabezpečení aplikací.
- **Aktualizace systému**, které přinášejí nové možnosti a bezpečnostní záplaty.
- **Ověření aplikací**, obsahující upozornění nebo automatické zablokování instalace škodlivých aplikací a průběžné skenování aplikací v zařízení.
- Systém detekce vniknutí **SafetyNet**, který pomáhá sledovat a snižovat známé bezpečnostní hrozby. Systém slouží mimo jiné i pro identifikaci nových bezpečnostních hrozeb.
- **Správce zařízení Android**, webová a mobilní aplikace vyhledávající ztracená nebo odcizená zařízení.

4.3 Architektura zabezpečení platformy

Android se snaží být nejbezpečnějším a nejpoužívanějším operačním systémem. Používané bezpečnostní modely se snaží mít co nejlepší, proto se mezi jejich hlavní cíle řadí: [11]

- ochrana uživatelských dat,
- ochrana všech systémových prostředků včetně ochrany sítě
- a vzájemná izolace všech aplikací.

K dosažení těchto cílů nabízí Android tyto klíčové funkce zabezpečení:

- robustní zabezpečení na úrovní OS pomocí Linuxového jádra,
- povinný Aplikační Sandbox (více k tomuto tématu v kapitole 4.3.1) pro všechny aplikace,
- bezpečná meziprocesová komunikace,
- poskytování oprávnění pro přístup k jednotlivým funkcím zařízení.

4.3.1 Aplikační Sandbox

Platforma OS Android využívá výhod linuxového zabezpečení uživatelů, a to hlavně identifikaci a izolaci zdrojů aplikací. Každá aplikace je spouštěna jako oddělený proces s unikátním uživatelským ID (též UID), které je přiřazováno OS Android. Takto dochází k izolaci a ochraně aplikací a systému před škodlivými aplikacemi.

O tuto funkci se na úrovni jádra stará Aplikační Sandbox. Jádro zajišťuje bezpečnost mezi aplikacemi a systémem na úrovni procesů, kdy jsou aplikacím nebo skupinám aplikací

přiřazována jedinečná ID, pomocí nichž mají dané aplikace uděleno povolení přístupu k určitým operacím. Aplikace ve výchozím nastavení nemohou vzájemně komunikovat a mají omezený přístup k operačnímu systému. Pokud se tedy aplikace A pokouší o neoprávněný přístup k aplikaci B (například čtení jejích dat), je jí to operačním systémem zakázáno.

Ačkoliv je Aplikační Sandbox uložen v jádře, rozšiřuje se i do nativního kódu a do aplikací operačního systému. Všechen software vystavěný nad Linuxovým jádrem, včetně knihoven operačního systému, aplikačního frameworku, aplikační runtime a všech aplikací, je spouštěn pomocí Aplikačního Sandboxu. Problémem jiných platforem je, že jsou vývojáři nuceni pracovat se speciálním vývojářským frameworkem nebo specifickým jazykem za účelem zvýšení bezpečnosti. Díky Sandboxu je totiž nativní kód stejně bezpečný, jako kód interpretovaný, a proto nemá OS Android žádná omezení toho, jak má být aplikace napsána.

Pokud v některých operačních systémech nastane porušení integrity paměti, jedná se o zásadní ohrožení bezpečnosti. V OS Android se tak neděje díky tomu, že všechny aplikace jsou na úrovni operačního systému kontrolovány Aplikačním Sandboxem. Chyba porušení integrity dovolí pouze spuštění libovolného kódu v rámci dané aplikace, která má povolena práva od OS Android.

Aplikační Sandbox stejně jako všechny ostatní bezpečnostní prvky OS Android není neprolomitelný. Aby jej však bylo možné obejít, bylo by (ve správně nakonfigurovaném zařízení) potřeba narušit bezpečnost Linuxového jádra. [12]

4.4 Zabezpečení systému a jádra

Na úrovni operačního systému poskytuje platforma Android zabezpečení jádra Linuxu i bezpečnou komunikaci mezi procesy (IPC), která umožňuje bezpečnou komunikaci mezi aplikacemi pracujícími v různých procesech. Tyto bezpečnostní prvky na úrovni operačního systému zajišťují, že i přirozený kód je omezen Aplikačním Sandboxem. Systém je navržen tak, aby zabránil škodlivým aplikacím poškozovat ostatní aplikace, systém Android nebo samotné zařízení. [13]

Linux Security

Základem platformy Android je Linuxové jádro, které má již řadu let širokou škálu využití a používá se v milionech prostředí citlivých na zabezpečení. Díky historii, do které nespadá jen neustálý výzkum a vývoj, ale také nejrůznější útoky a následné opravy tisíců vývojářů, se Linux stal stabilním a bezpečným jádrem, kterému důvěřuje mnoho korporací a odborníků na bezpečnost.

Systémový oddíl a nouzový režim

Systémový oddíl obsahuje nejen jádro OS Android, ale také knihovny operačního systému, aplikační runtime, aplikační framework a aplikace. Systémový oddíl je nastaven pouze ke čtení, nedá se do něj zapisovat. Pokud uživatel spustí zařízení v nouzovém režimu, jsou k dispozici pouze základní (předinstalované) aplikace. Uživatel tím zajistí, že se bude vyskytovat v prostředí

bez aplikací třetích stran, kde může zálohovat kontakty, fotky nebo jiná důležitá data. Může se stát, že se nám nepodaří zjistit, v které aplikaci se malware skrývá, a tak je praktické dostat se do prostředí, kde není žádná aplikace třetí strany.

Oprávnění souborového systému

V Unixovém prostředí zajišťuje oprávnění souborového systému to, že jeden uživatel nemůže měnit nebo číst ze souborů jiného uživatele. Podobně je tak v případě OS Android, kdy každá aplikace může číst pouze ze svých souborů a ne ze souborů jiné aplikace. Je pouze na vývojáři aplikace, zda k datům z aplikace bude mít přístup pouze aplikace samotná, nebo jestli cíleně zveřejňuje soubory jiným aplikacím a umožňuje jim tak přečíst nebo změnit soubory.

Kryptografie

Vývojářům je nabízena sada kryptografických rozhraní API, která mohou být využita v jejich aplikacích. Do této sady spadá implementace standardních a běžně používaných kryptografických funkcí, jako jsou AES (Advanced Encryption Standard), RSA (iniciály autorů Rivest, Shamir, Adleman), DSA (Digital Signature Algorithm) a SHA (Secure Hash Algorithm). Navíc jsou API poskytovány protokoly vyšší úrovně, jako jsou protokoly SSL (Secure Sockets Layer) a HTTPS (Hypertext Transfer Protocol Secure).

Root zařízení

Ve výchozím nastavení zařízení mají pouze jádro a malá část některých aplikací OS Android tzv. rootovská práva – práva superuživatele. Pokud uživatel nebo aplikace mají práva superuživatele, mohou měnit operační systém, jádro nebo jiné aplikace. Superuživatel má přístup ke všem aplikacím a do všech jejich dat. Nebezpečím při využití rootovských práv u zařízení je, že uživatel zařízení nejenže přichází o záruku zařízení, ale současně snižuje zabezpečení zařízení a vystavuje jej většímu množství potenciálních rizik.

Důvodem proč je root povolený a proč vůbec existuje, je jeho důležitost při vývoji aplikací na platformě OS Android. Díky bootloaderu, který mají uživatelé mnoha zařízení s OS Android možnost odemknout, je možné instalovat i jiné verze operačního systému. Tyto verze povolují uživateli získat root přístup za účelem ladění aplikací a systémových komponent nebo přístup k funkcím, které nejsou aplikacím běžně dostupné.

Jinou verzi operačního systému, který umožňuje změnit obyčejného uživatele na uživatele s rootovskými právy, může nainstalovat i uživatel s fyzickou kontrolou zařízení a USB kabelem. Aby byl původní uživatel i jeho data chráněna, jsou při procesu zpřístupnění bootloaderu smazána veškerá uživatelská data. Rootovská práva, která jsou získaná pomocí chyby jádra nebo bezpečnostní díry, mohou tuto ochranu obejít.

Šifrování dat pomocí klíče uloženého na zařízení neochrání data aplikace proti rootovským právům, jelikož uživatel s těmito právy má přístup všude, tedy i k šifrovacímu klíči. Aplikace (a vývojáři) mají možnost přidat vrstvu ochrany dat dané aplikace tím, že klíč uloží mimo samotné zařízení (na server nebo cloud). Tento přístup však tvoří jen dočasnou ochranu, jelikož pokud si

aplikace vyžádá přístup k datům, bude mít přístupný i klíč k rozšifrování dat, který se tímto způsobem dostane k danému klíči.

Robustnější přístup k ochraně dat od uživatelů s rootovskými právy je pomocí specializovaných hardwarových řešení. Výrobci zařízení využívající části hardwaru od jiných výrobců (tzv. OEM – Original Equipment Manufacturer) pak mohou omezit přístup k určitému obsahu zařízení využitím technických metod DRM (Digital Rights Management), jejichž účelem je ovládat či omezovat používání obsahu digitálních médií (hudba, videa, e-knihy atp.). Tohoto může být využito i k omezení přístupu k úložišti, které je využíváno technologií NFC (Near-field communication). Tato technologie umožňuje bezdrátovou komunikaci mezi elektronickými zařízeními na velmi krátkou vzdálenost (do 4 cm) a je využívána například jako Google peněženka (platební systém od společnosti Google).

Jelikož je klíč k dešifrování souborového systému chráněn uživatelským heslem, není možné v případě odcizení zařízení uživatelská data zpřístupnit. Použití tzv. zavaděče (z angličtiny bootloader), který běží před samotným spuštěním OS a může tím zabezpečit spuštění jiného OS, nebo úprava operačního systému, není v tomto případě dostatečným prostředkem pro přístup k uživatelským datům bez hesla uživatele. [13][14]

4.5 Zabezpečení uživatele

Šifrování souborového systému

Od Androidu verze 3.0 je poskytována možnost šifrování celého souborového systému, takže všechna uživatelská data včetně hesel mohou být šifrována na úrovni jádra. Šifrovací klíč je chráněn šifrou AES128, která používá na šifrování i dešifrování stejný klíč. Délky klíčů jsou 128, 192 nebo 256 bitů a nehrozí jim tedy útok hrubou silou (vyzkoušení všech kombinací hesel). V OS Android se tyto šifry používají s podporou klíče odvozeného z uživatelského hesla zadávaného při každém spuštění aplikace, což zabraňuje neoprávněnému přístupu k uloženým datům bez zadání uživatelského hesla pro dané zařízení.

Pro zaručení odolnosti vůči systematickým útokům na uhodnutí hesla (například výše zmíněnou hrubou silou) je heslo kombinováno s náhodnou „solí“. Kryptografická sůl je řetězec, který se určitou metodou přidává k heslu. Tato sůl je pokaždé jiná i pro stejná hesla, jelikož je generována náhodně. Po přiřazení soli k heslu je celý tento řetězec zašifrován pomocí některé z kryptografických funkcí. Takto zašifrované heslo je následně uloženo do databáze.

V Androidu verze 5.0 a vyšší je zabudována podpora šifrování celého disku. Heslo pro šifrování je chráněno heslem uživatele zařízení. Při startu zařízení a při každém dalším přístupu k datům uložených v zařízení je zapotřebí, aby uživatel zadal toto heslo pro zpřístupnění dané části disku.

Android verze 7.0 a vyšší podporuje šifrování na základě souborů. Šifrování založené na souborech umožňuje různým soubory šifrovat různými klíči a tyto soubory pak lze odemknout samostatně, nezávisle na ostatních. [15]

Ochrana heslem

Zařízení může požadovat po uživateli vytvoření hesla pro plný přístup k funkcím zařízení. Díky tomu lze v OS Android šifrovat klíč souborového systému a zamezit tak neoprávněnému užití přístroje. Výrobce zařízení může vytvoření a používání hesla vynutit a stejně tak může nastavit jeho minimální délku a složitost. [15]

4.6 Správa zařízení

OS Android verze 2.2 a novější obsahují podporu pro správu zařízení Android, která poskytuje funkce správy zařízení na systémové úrovni. Tyto funkce umožňují vytvářet aplikace podporující zabezpečení, které jsou užitečné v podnikových nastaveních, kde odborníci v oblasti IT vyžadují rozsáhlou kontrolu nad všemi zařízeními zaměstnanců. V podnikovém prostředí se často jedná o to, že zařízení zaměstnanců musí dodržovat přísný soubor zásad, kterými se řídí při používání zařízení. Jedná se tak například o kontrolu instalovaných aplikací či vynucování zásad zabezpečení zařízení.

Pokud uživatel zařízení s nainstalovanou aplikací správy zařízení nedodržuje specifikované zásady, pak tato aplikace může obsahovat algoritmus pro vynucení správného používání (aplikace tak může zakázat některé funkce, jako je synchronizace dat při nastavení hesla, které nesplňuje politiky podniku).

Administrátoři správy zařízení mohou požádat uživatele i o změnu hesla nebo uzamknout či vzdáleně vymazat data na ztraceném nebo odcizeném zařízení (tzn. obnovit výchozí tovární nastavení). [16]

Správa zařízení Android je kromě použití v aplikacích dodávaných se systémem Android dostupná také třetím stranám poskytovatelů řešení pro správu zařízení.

5 Přehled oprávnění Android aplikace

Účelem využívání oprávnění v OS Android je především ochrana soukromí uživatele tohoto systému. Vyvíjené aplikace musí z pravidla vyžadovat přístup k citlivým uživatelským údajům (jako jsou kontakty nebo SMS zprávy), nebo k některým funkcím systému (jako je využití fotoaparátu). V závislosti na této funkci může systém udělit žádané oprávnění automaticky nebo může požádat uživatele o schválení daného požadavku o udělení oprávnění.

Hlavním bodem návrhu architektury zabezpečení Android je skutečnost, že žádná aplikace nemá ve výchozím nastavení oprávnění provádět operace, které by měly nepříznivý vliv na ostatní aplikace, operační systém nebo data uživatele. To zahrnuje čtení a/nebo psaní soukromých dat uživatele (kontaktů nebo e-mailů), čtení a/nebo psaní dat jiných aplikací, provádění přístupu k síti, udržení přístroje v probuzeném stavu a podobně. [17]

5.1 Schválení oprávnění

Aplikace zveřejňuje potřebná oprávnění v tzv. manifestu, kde jsou do XML (eXtensible Markup Language) značek (tzv. tagů) `<uses-permission>` přidány názvy daných požadovaných oprávnění. Ukázka takového přidání oprávnění je zobrazena ve zdrojovém kódu v kapitole 6.6 (viz Zdrojový kód 3: Ukázka AndroidManifest.xml souboru).

5.1.1 Úrovně oprávnění

Existují tři hlavní úrovně, do kterých se oprávnění řadí a které ovlivňují aplikace třetích stran: běžné, podpisové a nebezpečné. Tyto úrovně ovlivňují především to, zda jsou vyžadovány požadavky na povolení oprávnění za běhu aplikace, či nikoliv.

Pokud jsou v manifestu aplikace uvedeny nějaké běžné oprávnění (tj. takové oprávnění, které nepředstavují velké riziko pro soukromí uživatele nebo provoz zařízení), jsou systémem automaticky udělena již při instalaci aplikace. Uživatel tyto oprávnění nemůže nijak modifikovat či rušit. Mezi běžné oprávnění se řadí mimo jiné i využívání internetu.

Jestliže však aplikace v seznamu zobrazuje některé z nebezpečných oprávnění (tj. takové oprávnění, která mohou potenciálně ovlivnit soukromí uživatele nebo normální provoz zařízení), musí uživatel výslovně souhlasit s udělením těchto oprávnění. Dokud uživatel neschválí povolení aplikaci používat dané oprávnění, nemůže aplikace poskytnout funkce závislé na tomto oprávnění. Tato nebezpečná oprávnění byla již nastíněna v kapitole 2, přesněji se jimi zabývá Tabulka 1: Nebezpečná oprávnění [1][2].

Jako podpisová oprávnění jsou pak uváděna taková oprávnění, při jejichž použití je nutný podpis certifikátu aplikace, která definuje dané oprávnění. Tato oprávnění často nejsou určena pro použití aplikacemi třetích stran, a proto se jimi nebudeme dále zabývat.

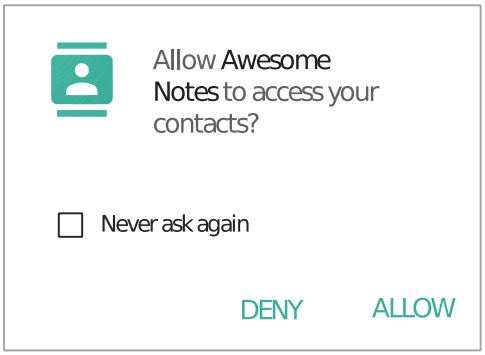
Existuje však ještě několik oprávnění, které jsou řazeny jak mezi běžné, tak mezi nebezpečné oprávnění (například oprávnění `WRITE_SETTINGS`). Tato oprávnění jsou označovány jako

obzvláště citlivé a většina aplikací by je vůbec neměla používat. Pokud aplikace potřebuje některé z těchto oprávnění, musí deklarovat oprávnění v manifestu a odeslat záměr vyžadující oprávnění uživatele. Systém reaguje na tento záměr zobrazením podrobnější zprávy uživateli.

5.1.2 Zobrazení požadavku o schválení

Pouze nebezpečné oprávnění vyžadují souhlas uživatele. Způsob, jakým OS Android požádá uživatele, aby udělil nebezpečné oprávnění, závisí na verzi systému Android spuštěném v zařízení uživatele a na verzi systému, na kterou se aplikace zaměřuje.

Pokud zařízení běží na systému Android 6.0 (úroveň 23 API) nebo vyšší, uživatel není v době instalace upozorněn na žádné oprávnění aplikace. Aplikace tak musí požádat uživatele o udělení nebezpečných oprávnění až za běhu resp. v případě potřeby. Když aplikace požádá o povolení, uživatel uvidí systémové dialogové okno, které uživatele informuje o tom, ke které skupině oprávnění bude mít aplikace přístup. Dialog z pravidla obsahuje tlačítko Odmítnout a Povolit. V případě, že uživatel odmítne zobrazený požadavek na schválení oprávnění, tak se při příštím zobrazení dialogu zobrazí i zaškrtávací políčko, kterým lze ovlivnit další výzvy k povolení oprávnění (viz Obrázek 2: Dialog s žádostí o udělení oprávnění).



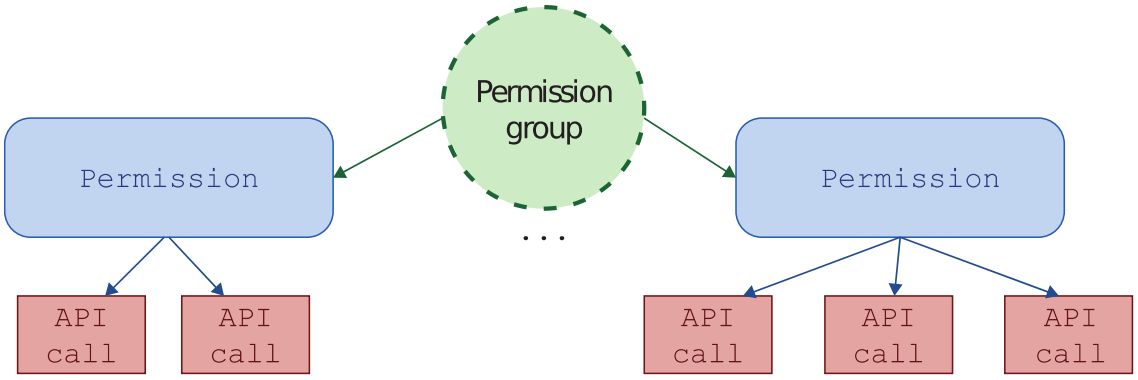
Obrázek 2: Dialog s žádostí o udělení oprávnění

Při využívání systému Android 5.1.1 nebo nižší, pak systém automaticky žádá uživatele o udělení všech nebezpečných oprávnění v době instalace. Pokud uživatel odmítne požadavek na oprávnění, pak systém zruší instalaci aplikace.

5.2 Skupiny oprávnění

Jak již bylo nastíněno v tabulce nebezpečných oprávnění (viz Tabulka 1: Nebezpečná oprávnění [1][2]), tak dochází k uspořádání oprávnění do skupin souvisejících s funkcemi zařízení. V OS Android jsou žádosti o povolení zpracovávány na úrovni skupiny a jedna skupina oprávnění odpovídá několika deklaracím oprávnění v manifestu aplikace (jak je znázorněno na Obrázek 3: Skupiny oprávnění [17]). Například skupina `SMS` pak obsahuje deklaraci `READ_SMS` a `RECEIVE_SMS` oprávnění, tedy oprávnění na čtení a příjem SMS zpráv.

Sdružování oprávnění tímto způsobem umožňuje uživateli provádět smysluplnější a informovanější volby, aniž by byl ohromen komplexními a technickými požadavky. Následně je důležité zmínit, že v případě, že aplikace vyžaduje oprávnění z dané skupiny oprávnění a v současné době nemá uděleno žádné oprávnění z této skupiny, tak je uživateli systémem zobrazeno dialogové okno s žádostí o danou skupinu oprávnění. Pokud však již dříve došlo k udělení jiného nebezpečného oprávnění ze stejné skupiny oprávnění, systém udělí požadované povolení bez jakékoli interakce s uživatelem. (Tento popis chování platí pouze pro OS Android 6.0 a vyšší.)



Obrázek 3: Skupiny oprávnění [17]

Všechny nebezpečné oprávnění jsou řazeny do skupin oprávnění, obecně však může každé oprávnění patřit do skupiny oprávnění bez ohledu na úroveň ochrany. Uživatelskou zkušenost (tzv. user experience) však ovlivňují pouze takové oprávnění, které jsou zařazeny do skupiny nebezpečných oprávnění.

5.3 Korektní používání oprávnění v projektu

Změny v modelu oprávnění vyžadují nemalé úsilí ze strany vývojáře aplikace, a to především v tom případě, že aplikace podporuje takové verze systému, kde tento model využívání oprávnění ještě neexistoval. Jelikož však moderní model oprávnění žádá o povolení přístupu k různým částem zařízení v různých stavech aplikace, je žádoucí zohlednit následující přístupy při vývoji aplikací. V případě, že by vývojář aplikace nezohlednil tyto přístupy, může se aplikace stát nestabilní v případě, že uživatel odebere některá oprávnění dané aplikace v nastavení zařízení. [18]

5.3.1 Zpětná kompatibilita

Při implementaci aplikace, která má být kompatibilní s verzemi OS Android nižší než verze 6.0, je nutné vzít v potaz, že v nižších verzích systému nebylo možné pracovat s oprávněními aplikace za běhu. Aplikace tak nepotřebuje ověření, zda uživatel udělil některé z nutných oprávnění pro správný běh aplikace, jelikož k udělení těchto oprávnění dochází již při instalaci aplikace. Proto v kritických místech aplikace, kde by v novém modelu docházelo k dotázání uživatele o povolení přístupu, je žádoucí kontrolovat verzi systému pomocí jednoduché podmínky zobrazené v následujícím zdrojovém kódu (viz Zdrojový kód 1: Kontrola verze systému). [18]

```
Build.VERSION.SDK_INT > Build.VERSION_CODES.LOLLIPOP_MR1
```

Zdrojový kód 1: Kontrola verze systému

5.3.2 Dotazování na povolení přístupu

OS Android ve verzi 6.0 a výše obsahuje funkci `requestPermissions(String[] permissions, int requestCode)`, která má za úkol zobrazení dialogového okna uživateli s žádostí o udělení přístupu k dané funkci zařízení. Do této funkce je pro zjednodušení možno

poslat několik názvů požadovaných oprávnění, pro které, pokud nejsou ze stejné skupiny oprávnění, dojde k vytvoření zmíněného dialogového okna. Na tuto metodu pak navazuje funkce `onRequestPermissionsResult(int permsRequestCode, String[] permissions, int[] grantResults)`, ve které je možno pracovat s výsledky požadavků o oprávnění. S těmito výsledky je nutné zacházet správným způsobem, a to tak, že v případě, kdy uživatel zamítne udělení oprávnění, je žádoucí zobrazit zprávu o tom, že některé funkce aplikace nebude možné využívat.

5.3.3 Nevytvářejte zbytečně otravné aplikace

Ať už je výsledek žádosti o udělení oprávnění jakýkoliv, nemělo by dojít k tomu, že aplikace bude znovu požadovat udělení stejného oprávnění v jiném případě, než kdy je to nutné z důvodu přístupu k dané funkci aplikace.

Níže uvedená metoda `checkSelfPermission(String permission)` by měla být využívána pokaždé, kdy aplikace vstupuje do místa, které vyžaduje použití nebezpečného oprávnění, a to i v případě, že již došlo k dřívějšímu udělení povolení (viz Zdrojový kód 2: Kontrola udělení oprávnění). Může se totiž stát, že uživatel toto povolení později odstranil pomocí nastavení zařízení.

```
if(checkSelfPermission(permission)==PackageManager.PERMISSION_GRANTED)
```

Zdrojový kód 2: Kontrola udělení oprávnění

Popisovaná metoda má návratovou hodnotu typu integer a může nabývat dvou stavů (`PERMISSION_GRANTED` nebo `PERMISSION_DENIED`). [18]

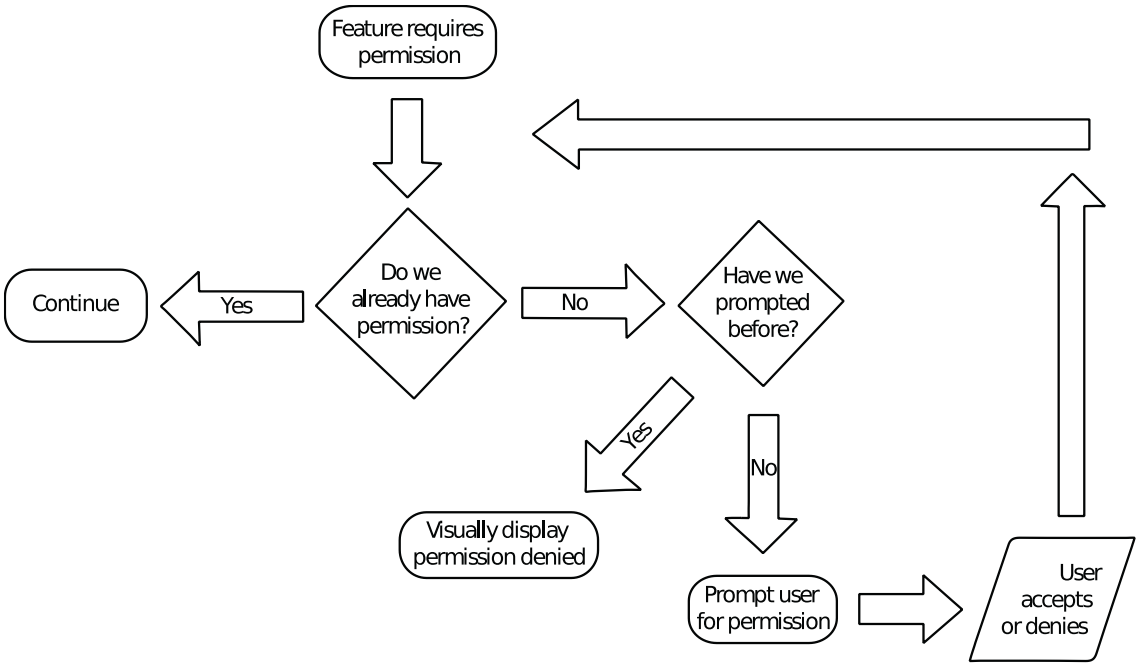
5.3.4 Neobtěžujte uživatele s dotazem na již udělené oprávnění

Existují případy, kdy je pro přístup k jedné funkci zařízení vyžadováno několik oprávnění (například u nahrávání videa jsou nutná oprávnění přístupu k fotoaparátu i k nahrávání zvuku, nehledě pak na přístup do paměti zařízení pro uložení nahrávky). Pokud již došlo k dřívějšímu přijetí některého z požadovaných oprávnění, tak bychom měli uživatele požádat pouze o ta oprávnění, o která zatím nebyla požádána.

Je vhodné podotknout, že to, že bylo o dané oprávnění dříve požádáno, neznamená, že bylo uživatelem uděleno. Znamená to jen to, že o něj bylo již dříve požádáno. Platí, že určení kolikrát vyzveme uživatele k udělení dříve odmítnutého oprávnění, je specifické pro danou aplikaci. Nikdy však není nutné vyzývat uživatele k povolení, které již bylo uděleno. [18]

5.3.5 Doporučený tok žádosti o oprávnění

Níže uvedený diagram (viz Obrázek 4: Diagram doporučeného toku žádostí o oprávnění [18]) vizuálně reprezentuje implementaci oprávnění popsanou v předchozích podkapitolách. Jedná se o návrh založený na oficiálních pokynech společnosti Google pro snížení opakovaných žádostí o udělení oprávnění na minimum.



Obrázek 4: Diagram doporučeného toku žádostí o oprávnění [18]

6 Komponenty Android aplikace

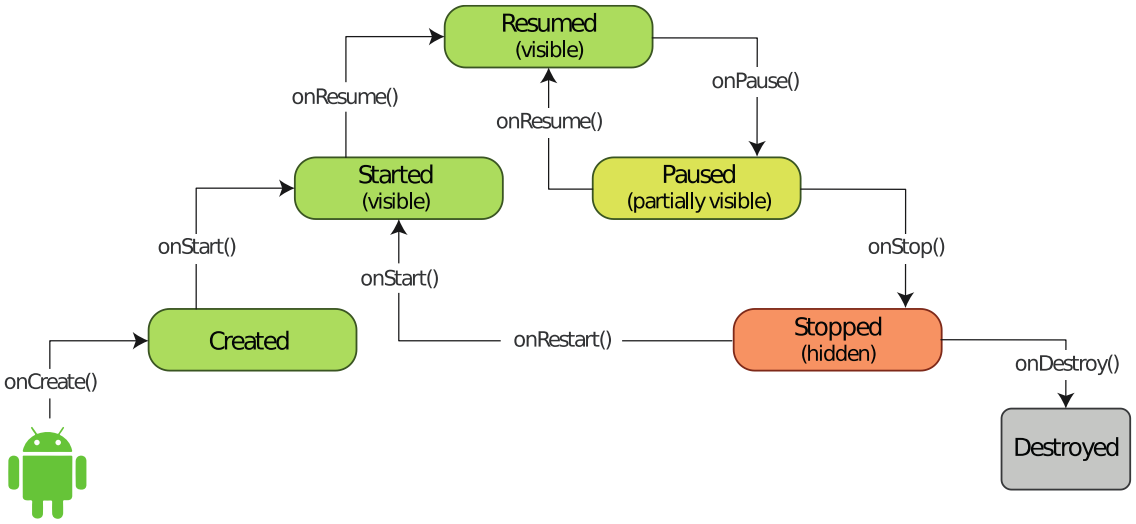
Základní stavební kameny v Android aplikacích jsou následující komponenty, kdy každá slouží k jinému účelu a má jiný životní cyklus definující jak je daná součást vytvořena a zničena. Aktivity reprezentující obrazovku, služby umožňující provádět akce na pozadí, poskytovatelé obsahu poskytující přístup k datům a přijímač vysílání reagující na příchozí oznámení. Každá tato součást aplikace může být vstupním bodem, kterým může systém nebo uživatel přistoupit k aplikaci. Některé komponenty jsou závislé na ostatních. Všechny tyto komponenty však musí být definovány v souboru AndroidManifest.xml, uloženém v kořenovém adresáři projektu. Kromě využití možností poskytovatelů obsahu mohou komponenty mezi sebou komunikovat pomocí zpráv, tzv. intentů. [19]

Jedinečným aspektem návrhu systému Android je, že každá aplikace může spustit komponenty jiné aplikace a ta po dokončení akce vrací získané hodnoty do původní aplikace pro následné použití. Vzhledem však k tomu, že systém spouští každou aplikaci v samostatném procesu, tak aplikace nemůže přímo aktivovat součást z jiné aplikace. Systém Android však může. Je-li žádoucí aktivovat součást z jiné aplikace, dochází k doručení zprávy, která určuje záměr spustit určitou součást, systému. Systém pak aktivuje komponentu namísto aplikace, která o to žádá.

6.1 Aktivita (Activity)

Aktivita je základním vstupním bodem určeným pro interakci s uživatelem. Představuje vždy jednu obrazovku s uživatelským rozhraním. Přestože aktivity společně tvoří soudržný uživatelský zážitek, každá z aktivit je nezávislá.

Činnosti aktivit usnadňují některé klíčové interakce mezi systémem a aplikací. Pomocí aktivit lze snadno sledovat, o co má uživatel zájem, neboli kterou aktivitu má spuštěnou, čímž je pak zajištěno, že systém bude nadále provozovat proces, který hostuje danou aktivitu. Dalším příkladem může být skutečnost, že předchozí procesy (ukončené aktivity) mohou obsahovat informace, ke kterým se uživatel bude chtít vrátit. Tím více je pak zapotřebí uchovávat tyto procesy (po nějakou dobu).



Obrázek 5: Životní cyklus aktivity [20]

Každá aktivita má svůj životní cyklus, který zobrazuje Obrázek 5. Obecně se aplikace může dostat do čtyř různých stavů. Aplikace může být:

- Aktivní (running), kdy došlo ke spuštění aplikace a daná aplikace běží v popředí (na obrázku se toto vztahuje na všechna tmavě zelená políčka).
- Pozastavená (paused), kdy došlo k narušení běhu aktivity upozorněním. Aktivita je stále spuštěná a je i částečně viditelná a čitelná, překrývá ji však okno s upozorněním (například oznámení o příchozí SMS zprávě).
- Zastavená (stopped), kdy došlo k zakrytí aktivity jinou aktivitou nebo jinou aplikací. Aktivita může stále komunikovat s uživatelem pomocí upozornění.
- Mrtvá nebo zničená (destroyed), kdy aktivita mohla být násilně ukončena například z důvodu nedostatku paměti v zařízení.

Spouštění a ukončování aktivit funguje na principu LIFO (Last-In-First-Out), kdy se nově spuštěná aktivita dostane na vrchol zásobníku spuštěných aktivit. Pokud uživatel právě spuštěnou aktivitu zavře nebo se vrátí o krok zpět, vždy se mu objeví aktivita dříve spuštěná. Tuto funkci lze ovšem přenastavit v rámci souboru AndroidManifest.xml nebo v rámci vytváření aktivity tak, aby nebyly uchovávány informace o předchozích aktivitách. [19]

6.2 Služba (Service)

Service, neboli služba je obecný vstupní bod pro udržování aplikace spuštěné v pozadí. Jedná se o součást, která běží na pozadí a provádí dlouhodobé operace nebo provádí práci pro vzdálené procesy. Služby však neposkytují uživatelské rozhraní. Služba může být využita například pro přehrávání hudby na pozadí, zatímco má uživatel aktivní jinou aplikaci a může sloužit i pro načítání dat přes síť (bez blokování interakce uživatele s aktivitou).

Existují dva velmi odlišné přístupy k využívání služeb, které udávají systému, jak spravovat danou aktivitu nebo aplikaci. Začaté služby říkají systému, aby je udržoval v chodu tak dlouho,

dokud nesplní požadovanou práci (například synchronizace dat). Vázané tzv. bound služby jsou spuštěny, protože některá jiná aplikace (nebo samotný systém) uvedla, že chce tuto službu využít. Jedná se v zásadě o službu poskytující rozhraní API jinému procesu. Systém tak ví, že existuje závislost mezi těmito procesy, takže jestliže proces A je vázán na službu v procesu B, ví, že musí udržovat proces B (a jeho službu) v operaci A. [19]

6.3 Poskytovatelé obsahu (Content provider)

Tzv. poskytovatel obsahu spravuje sdílenou sadu dat aplikací, do kterých mohou přistupovat různé aplikace. Tato data mohou být uložena v souborech, v databázi SQLite, na webu nebo na jiných trvalých úložištích a poskytovatel obsahu může aplikacím zamítnout přístup, pokud nemají příslušná oprávnění.

Poskytovatel obsahu je vstupním bodem do aplikace z pohledu systému. Poskytují mimo jiné přístup k čtení a zápisu soukromých a nesdílených dat aplikace. [19]

6.4 Přijímač vysílání (Broadcast receiver)

Přijímač vysílání je komponenta, která umožňuje systému dodávat události nebo zprávy do aplikace mimo běžný tok dat. Toto zaručí dané aplikaci možnost reagovat na celo-systémové události. Vzhledem k tomu, že přijímače broadcastů představují další dobře definovaný vstup do aplikace, může systém vysílat broadcasty i na aplikace, které v současné době nejsou spuštěny.

Mnoho broadcastů pochází přímo ze systému - broadcast oznamující vypnutí obrazovky, vybití baterie nebo zachycení snímku. I samotné aplikace však mohou iniciovat broadcast, a to pro předání zprávy ostatním aplikacím (například že došlo ke stažení nových dat do zařízení a že jsou k dispozici i pro jejich využití).

Přestože přijímače broadcastů nedisponují uživatelským rozhraním, mohou vytvořit upozornění ve stavovém řádku, které upozorní uživatele na danou událost. Obvykle je však přijímač broadcastů jen vstupní branou k jiným komponentám a je určen k velmi minimálnímu výkonu práce. [19]

6.5 Záměr (Intent)

Aplikace v OS Android mezi sebou komunikují pomocí tzv. intentů. Jsou to zprávy systému, které upozorňují sdružené aplikace, že se něco změnilo. Mezi takové změny patří například hardwarová konfigurace (např. nedostatek místa v paměti), upozornění na příchozí data (např. příchozí volání) nebo samotné události aplikace (např. selhání aplikace).

6.6 AndroidManifest.xml

Soubor AndroidManifest.xml je řídícím souborem psaným ve značkovacím jazyce XML, který popisuje celou aplikaci a její aktivity, služby, poskytovatele obsahu a záměry aplikace.

Tento soubor obsahuje mimo jiné i soupis všech oprávnění aplikace (sekce uses-permisions), která říkají, ke kterým funkcím zařízení může daná aplikace požadovat přístup. Ty si může uživatel přečíst v informacích o aplikaci v Google Play nebo v nastavení aplikací ve svém zařízení.

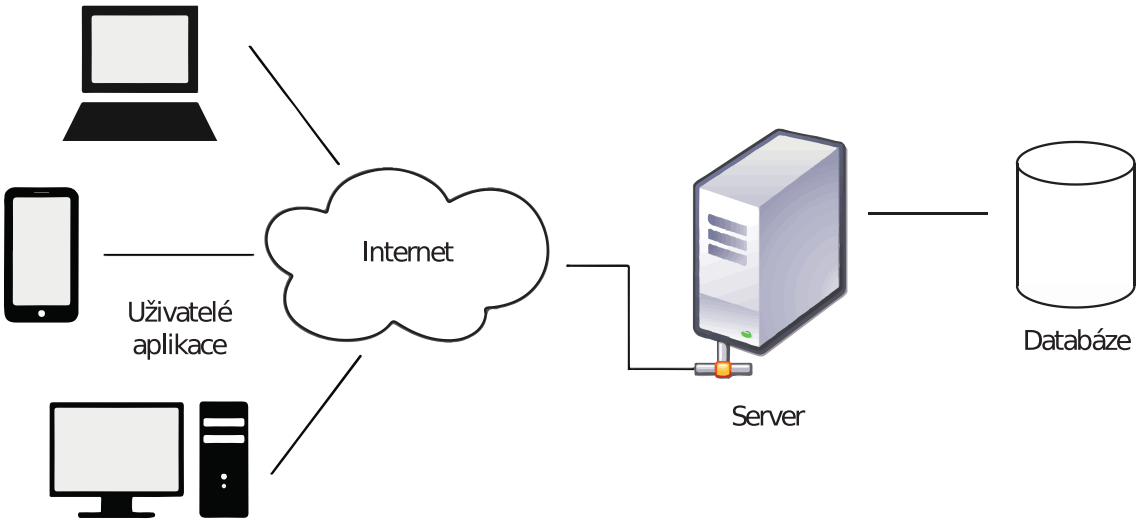
Součástí souboru AndroidManifest.xml je také název balíčku (package), ve kterém se skrývá celá aplikace. Aby mohla být aplikace distribuována v Google Play, musí být jméno balíčku jedinečné. Jméno balíčku se následně využívá jako název souboru Android Package, což je ZIP balík s koncovkou .apk, který se používá k instalaci a k šíření aplikace na zařízení s OS Android. Tento balík obsahuje zkomprimované zdrojové soubory aplikace.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myfirstapp">
    <uses-permission android:name="android.permission.INTERNET" />
        <application
           android:icon="@mipmap/ic_launcher"
           android:label="@string/app_name"
           android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Zdrojový kód 3: Ukázka AndroidManifest.xml souboru

7 Návrh aplikace

Při vytváření projektu došlo ke specifikaci základních vlastností aplikace, které byly následně v průběhu implementace (v některých případech) rozšiřovány. Pro vyjádření celkové specifikace implementovaného systému je zde uveden diagram popisující základní architekturu systému (viz Obrázek 6: Architektura projektu). Dle diagramu je zřejmé, že výsledná aplikace se skládá z několika částí, jejichž popisu se věnují následující kapitoly. Konkrétně se jedná o klientskou aplikaci, která byla vyvinuta pro OS Android a serverovou aplikaci, k níž je připojena databáze otestovaných aplikací.



Obrázek 6: Architektura projektu

Pro vyjádření celkové specifikace funkcí klientské aplikace bylo využito Use Case Diagramu, který umožňuje zobrazit chování aplikace v jazyce UML (Unified Modeling Language) tak, jak ho vidí uživatel. Use Case Diagramy obecně slouží převážně k popisu funkcionality systému, ovšem již dále nepopisují, jak je toho dosaženo. Jedná se tedy o sadu akcí, služeb a funkcí, které popisovaný systém provádí.

Pro popis datové struktury projektu je zde uvedena struktura databáze, a to za pomoci relačního datového modelu, který obsahuje základní informace o aplikacích a udělených oprávnění spolu s vypočteným skóre dané aplikace, které určuje bezpečnost aplikace.

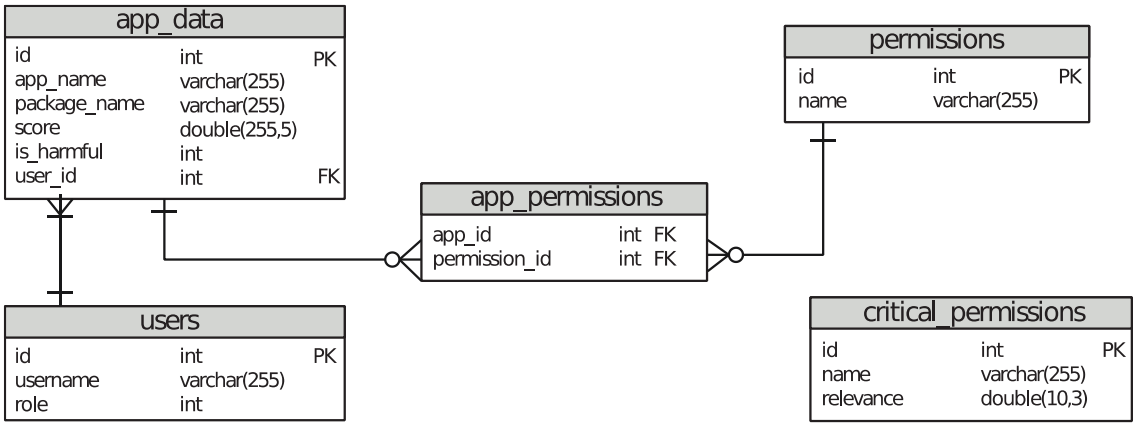
Následně jsou zde popsány klíčové prvky klientské a serverové části projektu. Mezi klíčové prvky klientské aplikace patří možnost oskenování celého zařízení na aplikace, které nejsou označovány jako systémové, nebo vybrání specifické aplikace a jejich následné ohodnocení. Jako klíčovou vlastnost je označena i možnost odinstalování dané aplikace po jejím ohodnocení v případě, že je její skóre vyšší než stanovená hranice. Mezi klíčové vlastnosti serverové části aplikace se mimo jiné řadí možnost kalkulace dříve zmíněného skóre aplikace, určujícího bezpečnost aplikace.

7.1 Databáze

V aplikaci je využíváno databáze, která obsahuje základní informace o testovaných aplikacích, převážně pak udělených oprávnění aplikace. Pro pochopení databáze a jejího obsahu je zde uveden relační datový model databáze použité v aplikaci (viz Obrázek 7: Relační datový model databáze). Tento model databáze má jednoduchou strukturu, kdy jsou data sdružována do tabulek neboli relací, které se skládají z řádků a sloupců. Následně jsou mezi těmito tabulkami zobrazeny vztahy, včetně jejich kardinality. Nad tabulkami uvedenými v relačním datovém modelu jsou pak prováděny veškeré databázové operace.

Struktura, která je použita v databázi, je následně převedena do datového modelu využívaného jak na serverové, tak na klientské části aplikace. Pomocí této struktury dochází k výměně důležitých dat mezi komponentami systému.

Kardinalita jednotlivých vztahů mezi tabulkami databáze je vcelku jasná z níže uvedeného relačního datového modelu (viz Obrázek 7: Relační datový model databáze), pro úplnost však musím dodat, že mezi tabulkami s názvy „app_data“ a „permissions“ existuje relace s kardinalitou M:N. Tento vztah umožňuje každému záznamu z jedné tabulky přiřadit libovolný počet záznamů z druhé tabulky. Dochází tak k vytvoření pomocné, tzv. vazební tabulky, ve které jsou zaznamenány kombinace primárních klíčů z těchto dvou zmíněných tabulek. V našem případě se pak jedná o tabulku „app_permissions“, která obsahuje pouze identifikátory, označené jako cizí klíče, propojených záznamů z tabulek „app_data“ a „permissions“. Následně je ještě vyznačena relace mezi tabulkami „users“ a „app_data“, jejichž kardinalita je 1:N.



Obrázek 7: Relační datový model databáze

V tabulce „permissions“ dochází ke sdružování záznamů o jednotlivých existujících oprávnění, které se vyskytují v testovaných aplikacích. Tato tabulka je spolu s tabulkou „app_data“ klíčová pro fungování celého systému, jelikož jsou v tabulce „app_data“ sdružována veškerá důležitá data o testovaných aplikacích (jako je i „package_name“ jehož zjištění je popsáno v kapitole 8.3.2, která se zabývá monitorováním udělených oprávnění aplikacím). Můžeme si povšimnout, že tato tabulka obsahuje sloupec s názvem „is_harmful“ s datovým typem integer. Tento sloupec je na klientské i serverové části aplikace implementován jako výčtový datový typ enum, který obsahuje hodnoty `TRUE`, `FALSE`, `CHECK` a `MARKASFALSE`.

Tyto hodnoty jsou následně využívány pro určení trénovací (hodnoty `TRUE` a `FALSE`) a testovací množiny (hodnota `CHECK`) při výpočtu skóre aplikace popsaném v kapitole 8.2.2.

Tabulka „users“ sdružuje data o uživatelích, kteří posílají své aplikace na otestování. Uživatelé mají v základu pouze „username“ neboli uživatelské jméno, ale v dalších rozšířeních aplikace je samozřejmě možné přidání hesla z pohledu autorizace a různé jiné vlastnosti. Pro tato možná rozšíření je přidána kolonka „role“, která je na klientské i serverové části implementována jako výčtový datový typ, díky kterému bude možné párovat zařízení uživatelů pro lepší kontrolu nad zařízeními.

Tabulka „critical_permissions“, zobrazena v relačním modelu (viz Obrázek 7: Relační datový model databáze) nemá vazby na žádnou tabulku, jelikož se jedná pouze o tabulku držící stálá data o existujících nebezpečných oprávněních, které byly uvedeny ve výše, resp. níže zmíněné tabulce (viz Tabulka 1: Nebezpečná oprávnění [1][2], resp. Tabulka 3: Nebezpečná oprávnění aplikací včetně jejich relevance). Položka „relevance“ v této tabulce byla určena dle závažnosti udělení daného oprávnění, jelikož některé oprávnění jsou více nebezpečné než jiné.

Postup při určování důležitosti neboli relevance daného oprávnění zobrazeném v následující tabulce (viz Tabulka 3: Nebezpečná oprávnění aplikací včetně jejich relevance), které může být v rozmezí 1 až 5, byl následující. Po nastudování, co přesně které oprávnění umožňuje na daném zařízení spravovat, je vcelku jasné, že některá z těchto oprávnění mají větší dopad na bezpečnost aplikace než jiná. Ty nejzávažnější oprávnění jsou pak ohodnoceny nejvyšší možnou známkou (v tabulce tedy číslem 5), zatímco ty nejméně závažné oprávnění mají ohodnocení nízké. Z praktického hlediska jsou nižším ohodnocením označena ta oprávnění, která jsou využívána aplikacemi považovanými za bezpečné (například aplikace od společnosti Google). Tato data jsou následně využívána při implementaci výpočtu skóre dané aplikace (viz kapitola 8.2.1).

Tabulka 3: Nebezpečná oprávnění aplikací včetně jejich relevance

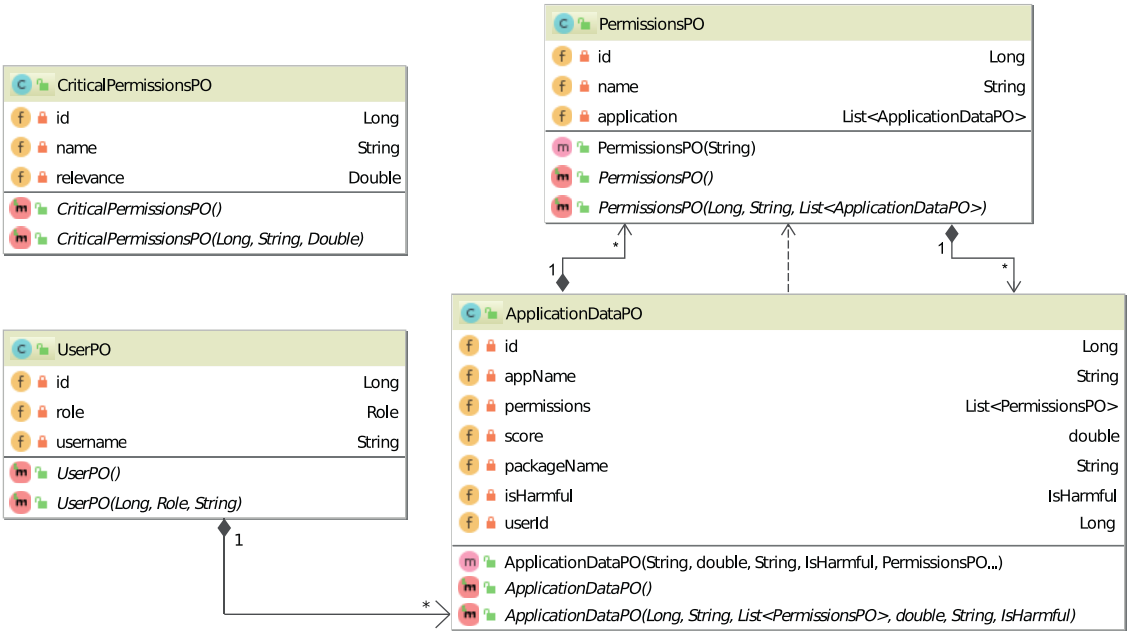
Název oprávnění	**Relevance (1-5)**	**Název oprávnění**	**Relevance (1-5)**
READ_CALENDAR	2	WRITE_EXTERNAL_STORAGE	3
WRITE_CALENDAR	4	READ_PHONE_STATE	5
READ_CALL_LOG	3	READ_PHONE_NUMBERS	4
WRITE_CALL_LOG	2	CALL_PHONE	4
PROCESS_OUTGOING_CALLS	4	ANSWER_PHONE_CALLS	5
CAMERA	3	ADD_VOICEMAIL	5
READ_CONTACTS	4	USE_SIP	4
WRITE_CONTACTS	3	BODY_SENSORS	3
GET_ACCOUNTS	5	SEND_SMS	5
ACCESS_FINE_LOCATION	4	RECEIVE_SMS	3
ACCESS_COARSE_LOCATION	4	READ_SMS	5
RECORD_AUDIO	4	RECEIVE_MMS	3
READ_EXTERNAL_STORAGE	3	RECEIVE_WAP_PUSH	3

7.2 Serverová část

Implementovaná aplikace využívá serverové části převážně pro získávání dat z výše popsané databáze již otestovaných a ohodnocených aplikací a pro ohodnocení zatím neotestované aplikace.

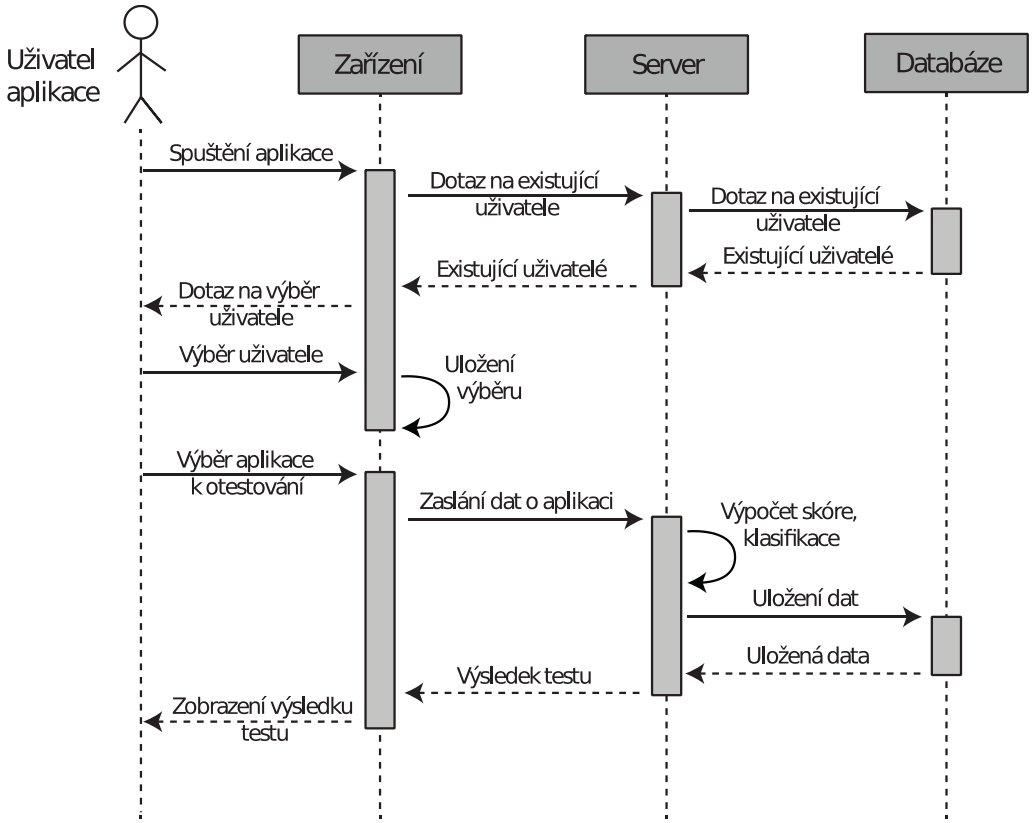
Níže zobrazený třídní diagram (viz Obrázek 8: Třídní diagram zobrazující klíčové třídy aplikace) popisuje zmíněnou klíčovou funkcionalitu serverové části aplikace. Jak lze na obrázku vidět, třídní diagram přesně odráží dříve zmíněnou strukturu využívané databáze.

Kardinalita vztahu M:N, která byla zmíněna v předchozí kapitole, je zde řešena pomocí datových struktur typu list, kdy v objektu znázorňující informace o aplikaci je list obsahující objekt udělených oprávnění a v oprávněních se vyskytuje list obsahující informace o aplikacích využívajících dané oprávnění.



Obrázek 8: Třídní diagram zobrazující klíčové třídy aplikace

Na následujícím obrázku (viz Obrázek 9: Sekvenční diagram aplikace) je zobrazena interakce mezi jednotlivými komponentami projektu, a to pomocí sekvenčního diagramu. Jak již název diagramu vypovídá, jedná se o zobrazení interakcí v takovém pořadí, v jakém se odehrávají. Tyto interakce v čase jsou v diagramu zobrazeny jako šipky od zdroje směrem k cíli požadavku. Díky této vlastnosti jsou diagramy vhodné pro zobrazení komunikace mezi objekty.



Obrázek 9: Sekvenční diagram aplikace

Sekvenční diagram popisuje klíčovou funkci aplikace, kterou je výpočet skóre aplikace a její následná klasifikace mezi potenciálně škodlivé nebo naopak neškodné aplikace. Na diagramu lze vidět, že hned po spuštění aplikace dochází ke kontaktování serveru (a databáze) na existující uživatele. Po vrácení existujících uživatelů z databáze dochází v klientské aplikaci k výběru uživatele, pod kterým budou prováděny všechny následující operace. Následně má uživatel možnost výběru aplikace, kterou chce otestovat. Po uživatelově výběru jsou data o vybrané aplikaci zaslána na server, kde proběhnou všechny výpočty spjaté s určením skóre a klasifikace. Tomuto následuje uložení dat do databáze a jejich vrácení zpět do klientské aplikace k prohlédnutí výsledků.

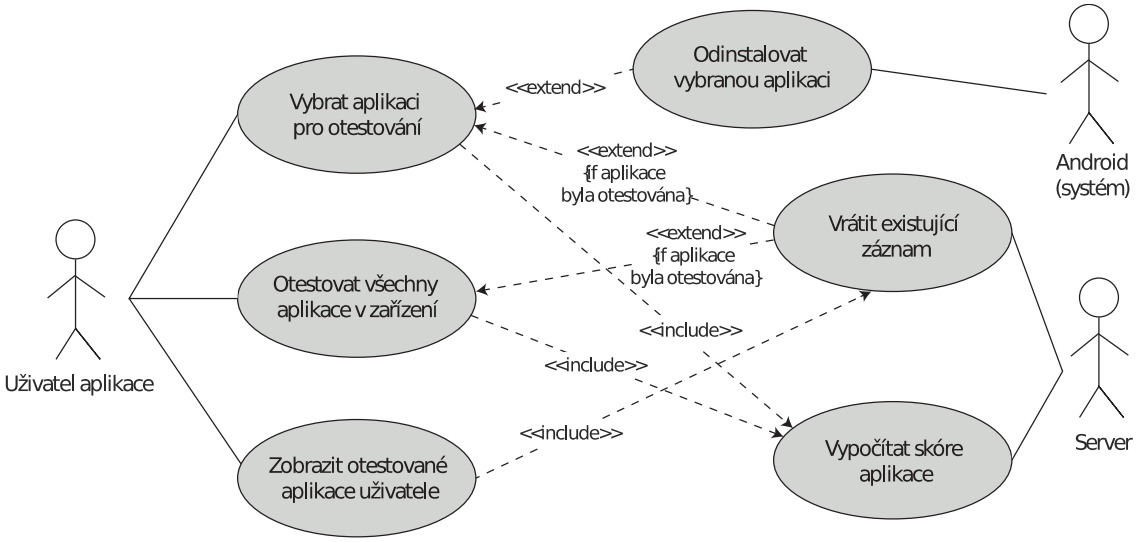
7.3 Klientská část

Aplikace byla navržena tak, aby byla splněna klíčová možnost otestování daných aplikací přímo ze zařízení, a to bez nutnosti využití tzv. rootu zařízení. Bylo tedy navrženo řešení s využitím Android aplikace, která umí detekovat ostatní nainstalované aplikace v zařízení. Požadavkem na tuto aplikaci bylo to, aby nebylo možné testovat systémové aplikace, jelikož takovéto aplikace ve většině případů není možné ze zařízení odinstalovat a mít tedy vypočtené jejich skóre by k ničemu užitečnému nevedlo.

7.3.1 Klíčové funkce klientské aplikace

Na následujícím obrázku (viz Obrázek 10: Use Case diagram zobrazující funkce aplikace), který znázorňuje diagram případu užití, jsou přehledně uvedeny klíčové implementované funkce, které jsou dostupné pro všechny uživatele aplikace.

Mezi tyto funkce patří především možnost otestování vybrané aplikace nebo možnost otestování všech aplikací nacházejících se v zařízení a následně možnost odinstalování celé aplikace. Aplikace navíc umožňuje uživateli vrátit veškeré záznamy z databáze týkajících se vybraného uživatele. Tímto způsobem je v aplikaci vyřešena možnost monitorování používaných oprávnění jiného uživatele. Toho je možné využít v následné implementaci dohledu nad jiným zařízením (například zařízením dítěte).



Obrázek 10: Use Case diagram zobrazující funkce aplikace

7.3.2 Uživatelské rozhraní aplikace

Jelikož operační systém Android běží na velké spoustě různě rozměrných zařízeních, bylo nutné navrhnout takové GUI (Graphical User Interface) aplikace, které bude jednoduché a intuitivní na všech velikostech zařízení a tedy i výsledné zobrazení aplikace bude vypadat přibližně stejně.

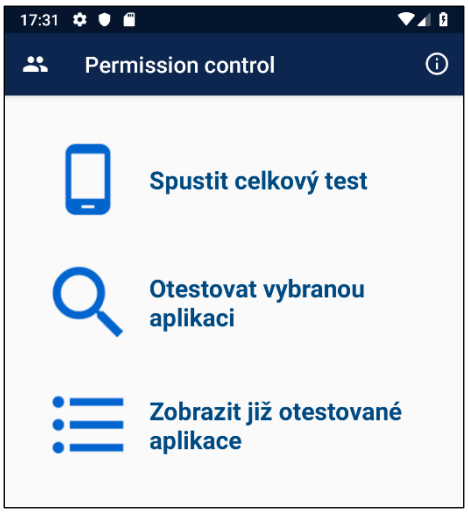
V aplikaci je zabudována podpora dvou jazykových modifikací, a to češtiny a angličtiny. K přepnutí mezi těmito jazyky je nutné využít nastavení zařízení, jelikož se vychází ze systémového nastavení.

Aplikace byla vyvíjena a testována na různých zařízeních, nejčastěji však na obrazovce s rozměrem 5“ a verzí Androidu 8.1.

Hlavní obrazovka aplikace

První obrazovku, kterou uživatel vidí po spuštění aplikace a po výběru uživatele, pod kterým se budou všechny následující operace ukládat do databáze, je zobrazena na následujícím obrázku

(viz Obrázek 11: Hlavní obrazovka aplikace). Dle obrázku je zřejmé, že uživatel má na výběr ze dvou možností, které jsou pro aplikaci klíčové.



Obrázek 11: Hlavní obrazovka aplikace

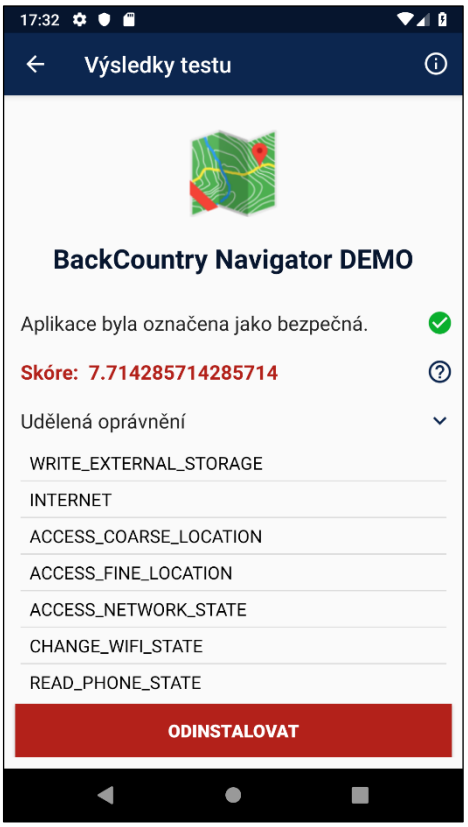
Po vybrání kterékoli z těchto možností dojde k zobrazení dialogu pro výběr následné akce, ať už se jedná o potvrzení zvolené akce, či výběr aplikace pro otestování.

Třetí možností je zobrazit všechny již otestované aplikace a jejich výsledky, a to nejen aplikace otestované na tomto zařízení a tímto uživatelem, ale i otestované aplikace jiných uživatelů uložených v databázi. Touto funkcí je umožněn jednoduchý dohled nad tím, jaké aplikace a jejich oprávnění jsou využívána například na zařízení dítěte.

Aplikace obsahuje srozumitelný návod na obsluhu, skrytý pod ikonkou informací v menu, které je umístěno v horní části obrazovky a je přístupný ze všech aktivit aplikace.

Výsledky testu

Po zvolení jedné z možností z výchozí obrazovky aplikace je uživatel přesměrován do druhé aktivity aplikace, kde jsou mu přehledně zobrazeny výsledky testování aplikace. Uživatel zde mimo základních informací o testované aplikaci (jako je používaná ikonka dané aplikace a její název) vidí to nejdůležitější z celé aplikace, což je vypočtené skóre. Toto skóre pak může být zobrazeno ve dvou barvách, které zcela intuitivně ukazují, zda je aplikace potenciálně nebezpečná (červená barva), či nikoli (zelená barva). Uživateli je k těmto barvám přidána vysvětlivka, která je umístěna přímo v aplikaci (skryta pod symbolem otazníku). Následně jsou zde přehledně zobrazena všechna udělená oprávnění vybrané aplikace, které je možné schovat pomocí šipky směřující dolů.



Obrázek 12: Zobrazení výsledku testu aplikace

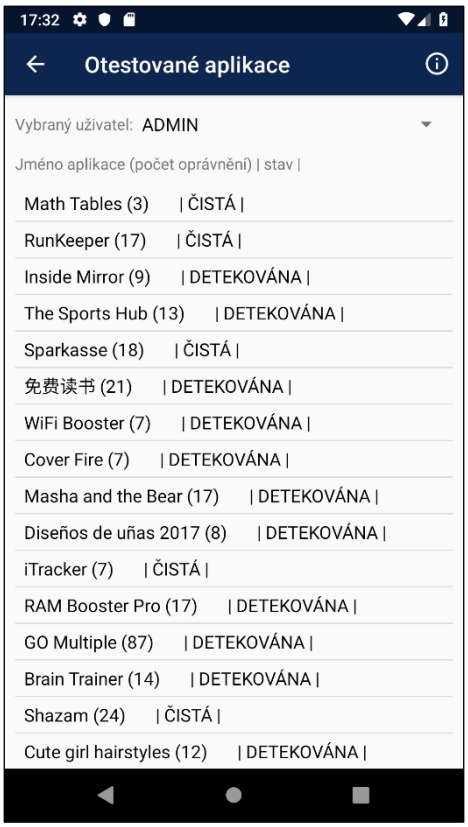
Výsledek testu obsahuje i zobrazení výsledné klasifikace aplikace, jejíž algoritmus je vysvětlen v kapitole 8.2.4. V případě, že je aplikace klasifikována jako bezpečná, zobrazí se uživateli informace označená zeleným háčkem. Pokud je aplikace označena jako potenciálně nebezpečná, je uživateli zobrazena výrazná zpráva informujícího ho o tomto stavu, následována ikonou vykřičníku. Uživatel má následně možnost toto tvrzení upravit pomocí zaškrtávacího políčka, které danou aplikaci označí jako důvěryhodnou a neškodnou.

Ať už je výsledek skóre nebo klasifikace pozitivní či negativní, je uživateli umožněno vybranou aplikaci odebrat respektive odinstalovat ze zařízení. Příklad možného výsledku testu aplikace je zobrazen na předchozím obrázku (viz Obrázek 12: Zobrazení výsledku testu aplikace).

Zobrazení testovaných aplikací podle vybraného uživatele

Jak již bylo zmíněno výše, uživatel aplikace má možnost podívat se na všechny testované aplikace, a to nejen ty, které testoval on sám, ale i na ty, které byly otestovány někým jiným na jiném zařízení. K tomuto slouží aktivita „Otestované aplikace“, ve které lze vybrat uživatelské jméno, podle kterého následně dojde k vylistování jeho otestovaných aplikací.

V seznamu nejsou zobrazeny pouze názvy aplikací, ale jak napovídá nápověda nad seznamem, obsahuje také počet udělených oprávnění dané aplikaci a její klasifikaci označenou jako stav. Dalo by se očekávat, že čím více má aplikace oprávnění, tím spíš bude zařazena mezi potenciálně nebezpečné aplikace. Jak lze na obrázku (viz Obrázek 13: Zobrazení otestovaných aplikací) vidět, nemusí to nutně být pravda.



Obrázek 13: Zobrazení otestovaných aplikací

8 Implementace aplikace

Tato část práce se zaměřuje na detailní popis softwarové části aplikace. Vzhledem k tomu, že implementace aplikace byla rozdělena na dvě hlavní softwarové části, a to na klientskou část a část serverovou spolu s databází, budou zde popsány pouze třídy, či funkce, které nejsou triviální.

8.1 Databáze

Vytvoření struktury databáze bylo provedeno pomocí anotací využitých v POJO (Plain Old Java Object) datových třídách serverové části aplikace pomocí standardu JPA (Java Persistence API), který byl importován pomocí balíčku javax.persistence, který popisuje programátorské rozhraní a chování knihoven pro objektově-relační mapovaní (ORM – Object-Relational Mapping) [21]. Pro správnou implementaci JPA bylo využito frameworku Hibernate[1].

Na následující ukázce zdrojového kódu (viz Zdrojový kód 4: Využití anotací při implementaci databáze) lze vidět, že pro správné vytvoření tabulky v databázi je potřebné využití anotace `@Entity`, kde je možné (nikoliv nutné) specifikovat jméno tabulky. Následně je pro správné indexování položek v tabulce potřebné využití anotací `@Id` a `@GeneratedValue`. Tyto anotace nám zajistí generování identifikátorů pro každý záznam v tabulce. Pro každý sloupec je dále možné využití anotace `@Column`, kde je opět možné specifikovat jméno daného sloupce a jeho další vlastnosti, jako je délka záznamu nebo zda může být tento sloupec ponechán prázdný či nikoliv. Pro potřebu implementace kardinality vztahu M:N bylo ještě využito anotací `@ManyToMany` a `@JoinTable`.

```
@Entity(name = "app_data")
public class ApplicationDataPO implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "app_name", nullable = false, unique = true)
    private String appName;

    @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JsonIgnoreProperties("application")
    @JoinTable( name = "app_permissions", joinColumns =
       { @JoinColumn(name = "app_id", referencedColumnName = "id")},
       inverseJoinColumns = {@JoinColumn(name = "permission_id",
       referencedColumnName = "id")})
    private List<PermissionsPO> permissions;
    ...
}
```

Zdrojový kód 4: Využití anotací při implementaci databáze

[1] Hibernate poskytuje způsob, jakým lze zachovat stav objektů v průběhu aplikace. Toho je dosaženo pomocí mapování objektů na entity v relační databázi (zkratka ORM). Mapování je umožněno dvěma způsoby – pomocí mapovacích souborů nebo pomocí anotací. [22]

Jako objektově-relační databázový systém byl zvolen PostgreSQL, který je vydáván pod MIT licencí (licence vznikla na Massachusettském technologickém institutu). Jedná se tak o software, který je zdarma a je tzv. open-source. Jedná se o databázový systém, který používá a rozšiřuje jazyk SQL (Structured Query Language) s mnoha funkcemi, které bezpečně ukládají a rozšiřují nejsložitější pracovní úlohy [23].

Administračním rozhraním využívaným pro správu této databáze byl open-source program s názvem pgAdmin 4, a to ve verzi 3.0. Tento program poskytuje grafické rozhraní pro využívaný databázový systém PostgreSQL. [24]

8.2 Serverová část

Pro implementaci serverové části aplikace bylo využito IDE IntelliJ IDEA verze 2018.3.4 a programovacího jazyka Java ve verzi 11. Na serveru je využíváno Spring Boot rámce (tzv. frameworku) ve verzi 2.1.3. Využití tohoto rámce poskytuje vývojáři rychlý a komplexní způsob vytváření samostatných (tzv. stand-alone) aplikací a způsob vývoje mikroslužeb (z angličtiny microservices). Na straně serveru pak dochází především ke komunikaci s vytvořenou a dříve popsanou databází a k výpočtu skóre aplikace a ke klasifikaci závisející na udělených oprávněních.

Po spuštění serverové části aplikace dojde ke spuštění serveru na tzv. localhostu, tedy na právě používaném počítači, a to na portu 3333, který je specifikován v souboru s názvem application.yml. Při tomto spuštění dochází k inicializaci nebo k připojení k využívané databázi, a to v závislosti na nastavení uvedeném v souboru application.yml. V tomto souboru je mimo jiné uvedeno, na jaké adrese je očekáváno spuštění databázového serveru (v našem případě se jedná opět o localhost, port 5432).

8.2.1 Přístupové body serveru (tzv. endpointy)

Pro možnost kontaktování serveru ať již pro zobrazení výsledku otestované aplikace nebo pro zaslání nových dat pro výpočet skóre vybrané aplikace bylo nutné vytvoření takových endpointů na serveru, které budou vracet očekávaná data. Endpoint neboli přístupový bod webové služby je adresa URL (Uniform Resource Locator), díky které se klienti mohou dostat k operacím poskytovaným touto službou. Přístupové body jsou implementovány jako třídy označovány anotací `@RestController`.

Endpointy v aplikaci navíc využívají anotace `@PostMapping` nebo `@GetMapping`, které určují, jaká metoda je využita při HTTP (Hypertext Transfer Protocol) dotazu. V následujícím seznamu jsou stručně popsány přístupové body serveru včetně jejich konkrétní cesty pro mapování HTTP požadavků a metody využívané v HTTP dotazu:

- **GET /appData** – Slouží k získání seznamu všech existujících záznamů z databáze nacházejících se v tabulce „app_data“. Tyto záznamy obsahují všechna data o již otestovaných aplikacích včetně seznamu jim udělených oprávnění.
- **GET /appData/{appName}/{userId}** – Podle zadaného ID uživatele a názvu aplikace dochází k výběru a zaslání dat z daného záznamu z databáze. Tyto dvě

položky mají tu vlastnost, že jejich kombinací vzniká unikátní klíč sloužící pro výběr informací o jedné specifické aplikaci.

- **POST /appData/save** – Metoda, která je použita po přístupu k tomuto přístupovému bodu slouží k uložení zaslaného objektu do databáze, přesněji do tabulek „app_data“ a „permissions“. Před samotným uložením objektu dochází k výpočtu skóre aplikace a také k její klasifikaci mezi potenciálně škodlivé či neškodné aplikace. Metoda následně vrací tento vytvořený záznam z databáze, a to včetně všech nových informací přidaných k tomuto objektu.
- **POST /appData/multiEval** – Zavolání tohoto přístupového bodu dochází při zasílání většího množství aplikací k ohodnocení. Po provedení výpočtu skóre a klasifikace dochází k uložení těchto nových záznamů do databáze a následně k vrácení seznamu všech takto zaslaných a ohodnocených objektů.
- **GET /appData/user/{userId}** – Slouží k získání všech otestovaných aplikací dle ID vybraného uživatele.
- **GET /permissions** – Slouží k získání všech existujících záznamů z databáze týkajících se oprávnění udělovaných aplikacím (tabulka „permissions“).
- **GET /permissions/{name}** – Metoda vrací jeden záznam z databáze podle zadaného názvu oprávnění. Toto je možné díky tomu, že databáze obsahuje pouze unikátní názvy existujících oprávnění.
- **GET /criticalPermissions** – Metoda slouží k získání všech záznamů z databáze týkajících se kritických neboli nebezpečných oprávnění uložených v tabulce „critical_permissions“.
- **GET /criticalPermissions/{id}** – Po zavolání tohoto přístupového bodu dochází k vrácení jednoho záznamu z tabulky nebezpečných oprávnění („critical_permissions“) určeného dle zadaného ID tohoto nebezpečného oprávnění.
- **GET /criticalPermissions/{name}** – Po zavolání dochází k vrácení jednoho záznamu z tabulky „critical_permissions“, a to podle zadaného názvu nebezpečného oprávnění. Toho je možné docílit z důvodu unikátnosti názvů jednotlivých oprávnění.
- **GET /users** – Metoda slouží k získání seznamu všech existujících uživatelů nacházejících se v tabulce „users“.
- **GET /users/{id}** - Po zavolání tohoto přístupového bodu dochází k vrácení jednoho záznamu z tabulky uživatelů specifikovaného dle zaslaného ID uživatele.
- **POST /users/save** – Metoda slouží k uložení nového uživatele specifikovaného zaslaným objektem. Po uložení dochází k vrácení uloženého záznamu.

Komunikace mezi serverem a klientem odpovídá charakteru REST (Representational State Transfer) neboli RESTful API, které využívá HTTP požadavků GET, PUT, POST a DELETE pro operace s daty (získání, poslání a mazání dat). REST API, které je také označované jako RESTful webová služba, tvoří architekturu rozhraní, která je založena na technologii reprezentativního přenosu stavu. Jedná se tak o přístup ke komunikaci mezi klientem a serverem často využívaným ve vývoji webových služeb.

8.2.2 Služby serveru

Služby jsou součásti servisní vrstvy, které stanovují množinu dostupných operací v rámci komunikace se serverovou částí. Služby jsou implementovány jako třídy a jsou označovány anotací `@Service`. Každá z těchto tříd obsahuje funkce, které jsou specifické pro požadované úlohy. V aplikaci jsou tyto funkce volány z dříve zmiňovaných přístupových bodů nebo z jiných služeb.

Z hlediska logické struktury tříd jsou tyto služby rozděleny do několika částí, kdy každá z nich obsahuje metody specifické pro danou část aplikace, kterou zastupují. Aplikace obsahuje služby starající se o přístup k aplikačním datům a k uloženým oprávněním aplikací, dále také přístup k nebezpečným oprávněním a k informacím vztahujícím se k uživateli aplikace.

Služba starající se o aplikační data obsahuje mimo jiné i klíčové metody aplikace, jako je výpočet skóre aplikace nebo její klasifikace, které jsou popsány v následujících podkapitolách. Stěžejní metodou v této službě bylo řešení ukládání nových dat, respektive jejich aktualizace v případě, že došlo ke změně udělených oprávnění nebo k označení důvěryhodnosti aplikace uživatelem. Jak již bylo zmíněno dříve, mezi tabulkami udržující informace o aplikacích a o udělených oprávněních existuje vztah s kardinalitou M:N. Z tohoto důvodu bylo jednodušším řešením tato data při jejich aktualizaci smazat a vytvořit nová, protože kvůli využití anotací pro vytvoření tohoto vztahu nedošlo k vytvoření vazební tabulky pomocí POJO tříd a tudíž k této tabulce nebyl umožněn jednoduchý přístup.

8.2.3 Výpočet skóre aplikace

Klíčovým prvkem aplikace je výpočet skóre aplikace založeném na ohodnocení udělených oprávnění dané aplikace. Pro tento výpočet bylo využito dříve zmíněných nebezpečných oprávnění a jejich ohodnocení (viz Tabulka 3: Nebezpečná oprávnění aplikací včetně jejich relevance), a to dle níže popsané rovnice (viz [25]).

$$RII = \frac{\Sigma a_i w_i}{AN} = \frac{5n_5 + 4n_4 + 3n_3 + 2n_2 + 1n_1}{5N} \quad (1)$$

Tato rovnice (1) udává, že pro posouzení skóre aplikace a relevance významu udělených nebezpečných oprávnění, bylo využito rovnice *RII*, tedy výpočtu Relative Importance Index [25]. Rovnice RII pomáhá se zjištěním podílu důležitosti dané proměnné v datech. Pro správnost výpočtu je zapotřebí sečíst všechny relevance udělených nebezpečných oprávnění (v rovnici zobrazeno jako a_i w_i). Následně je tento výsledek nutno vydělit hodnotou násobku nejvyšší možné relevance (v našem případě se $A = 5$) a celkovému počtu udělených nebezpečných oprávnění (v rovnici zobrazeno jako N). Platí, že čím vyšší skóre *RII*, tím důležitější jsou využité faktory pro výpočet. [25]

Po vypočtení skóre aplikace pomocí této rovnice dostaneme desetinné číslo v rozmezí od 0 do 1, pro naše požadavky výsledku skóre v rozsahu od 0 do 10 je proto nutné ještě celý výsledek přenásobit konstantou 10.

8.2.4 Klasifikace aplikace

Klasifikace aplikace je ukládána v již dříve zmíněném výčtovém datovém typu IsHarmful, který obsahuje hodnoty `TRUE`, `FALSE`, `CHECK` a `MARKASFALSE`. Jestliže je v tomto datovém typu uložena hodnota `CHECK`, dochází v algoritmu k určení klasifikace, která určuje, zda je aplikace škodlivá (následně uložena jako hodnota `TRUE`) nebo nikoliv (hodnota `FALSE`). Hodnota `MARKASFALSE` pak slouží pouze pro uživatelovo určení, že testovaná aplikace je dle něj bezpečná.

S klasifikací vybrané aplikace souvisí využití algoritmu s názvem k-Nearest Neighbors dále označovaným jako kNN. Tento algoritmus je jedním z nejjednodušších algoritmů využívaných pro klasifikaci dat a je jedním z nejpoužívanějších učících algoritmů [26]. Pro využití kNN je vhodné použití databáze, ve které jsou data rozdělena do několika (v našem případě dvou) tříd. Tato data a jejich rozdělení pak slouží k předvídání třídy nového vzorku dat a tím jeho klasifikace do již vytvořených tříd. Nový prvek je klasifikován podle „hlasování“ sousedních prvků – algoritmus specifikuje *k* nejbližších prvků (nejčastěji je *k = 3* nebo *5*), podle jejichž vlastností určí, do jaké třídy bude patřit nový prvek. [26]

Jelikož je struktura výsledného modelu klasifikace určena jen ze vstupních dat, je vhodné využít algoritmu kNN především v případech, kdy nemáme žádné nebo máme jen velmi malé znalosti o distribuci těchto vstupních dat.

V našem konkrétním případě algoritmus postupuje tak, že dojde k vytvoření vektorů dat z udělených nebezpečných oprávnění aplikací uložených v databázi. K tomu dojde pomocí kontroly, zda se udělené oprávnění vyskytuje v datech z tabulky „critical_permissions“ udržující záznamy nebezpečných oprávnění (viz Obrázek 7: Relační datový model databáze), či nikoli. Tyto vektory dat z jednotlivých aplikací byly následně využity pro vytvoření matice velikosti *počet_aplikací X počet_nebezpečných_oprávnění*. V databázi byla v době vytváření této matice umístěna pouze data takových aplikací, u kterých bylo dle zdroje [27] zjištěno, že se jedná o škodlivé aplikace, nebo takové aplikace, které již byly klasifikovány dříve.

Následně dochází k určení klasifikace nové aplikace v závislosti na využívaných oprávněních. Ze všeho nejdříve bylo nutné nalezení *k* specifikovaných sousedů, kteří obsahují co nejvíce stejných oprávnění. Pro hledání nejbližších sousedů lze použít různé metriky, kdy nejobvyklejší je euklidovská metrika, která byla v našem případě využita (viz následující rovnice).

$$m_e(\vec{a}, \vec{b}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{2}$$

Vektory *a* a *b* označují vektory o stejné velikosti a reálně se jedná o vektory zobrazující jak nově klasifikovanou aplikaci, tak jednu z již ohodnocených aplikací. Algoritmus postupně prochází všechny již ohodnocené aplikace a ke každé z nich přiřazuje vybranou aplikaci určenou ke klasifikaci. Výsledky této rovnice jsou při každém průchodu uloženy v datové struktuře spolu s ohodnocením dané aplikace a po provedení výpočtu pro všechny již ohodnocené aplikace dojde

k vybrání *k* nejnižších hodnot. Pro klasifikaci nové aplikace je použito to ohodnocení, které se v těchto nejnižších hodnotách vyskytuje častěji.

8.3 Klientská část

Pro implementaci klientské části aplikace bylo využito IDE Android Studio verze 3.3.1 s přidanou podporou pro programovací jazyk Kotlin, ve kterém je celá aplikace napsaná. Na straně klienta dochází převážně k vizualizaci dat obsažených v databázi a k získávání důležitých informací o nainstalovaných aplikacích v zařízení.

8.3.1 Verze SDK aplikace

Pro správné fungování jedné z klíčových funkcí aplikace bylo nutné specifikovat minimální podporovanou verzi Android SDK na verzi 23, tedy Android verze 6.0 s označením Marshmallow. Zařízení s nižší verzí OS Android tedy nebudou moct využít funkcí této aplikace.

Specifikace právě této verze byla nutná z jednoho prostého důvodu, a tedy, že až od této verze OS Android je možné spravovat udělená oprávnění tzv. za běhu. Tento model poskytuje uživatelům systému verze 6.0 a vyšší lepší viditelnost a kontrolu nad oprávněními a současně zjednodušuje proces instalace aplikace. Uživatelům je tedy poskytnuta důležitá možnost volby při udělování (a případnému následnému odvolávání) vybraných oprávnění, a to jednotlivě pro nainstalované aplikace [28]. Naopak v předchozích verzích tohoto systému byla uživateli poskytnuta jediná možnost, a to udělit požadované oprávnění aplikaci hned při instalaci. Tato oprávnění již následně nebylo možné nijak upravovat.

Stěžejní součástí při vývoji aplikace však může být komplikovanější přístup k zjišťování, zda má daná aplikace udělené dané oprávnění či nikoliv (a v tomto případě vyzvat uživatele k udělení daného oprávnění). Vývojářům aplikací na OS Android tak nezbývá nic jiného, než zajistit správný přístup k tomuto modelu, a to za pomoci podmínek typu „jestliže – pak“ v případě využívání funkcí aplikace vyžadujících oprávnění. Tento postup je pak nutný využívat ve všech verzích SDK, tedy i nižší než verzi 23.

Použitou verzi SDK v aplikaci lze nalézt v souboru s názvem `build.gradle`, který se nachází v adresáři „app“ vytvořeného projektu. V tomto souboru je specifikována nejen minimální podporovaná verze SDK (`minSdkVersion`), ale také jaká je výchozí verze pro kompilaci (`compileSdkVersion`) a na jaké nejvyšší verzi byla aplikace testována (`targetSdkVersion`). Tento soubor může mimo jiné obsahovat nutné závislosti (tzv. `dependencies`) pro využívání externích knihoven nebo specifikaci využívaných zásuvných modulů (z angličtiny „plugin“) v projektu.

8.3.2 Monitorování využívaných oprávnění aplikací

Pro možnost ohodnocení vybrané aplikace bylo klíčové zjistit, jaké oprávnění jsou v dané aplikaci povolené. K tomuto slouží především metoda `getPermissionByPackageName()`,

která je popsána v následujícím zdrojovém kódu (viz Zdrojový kód 5: Metody pro zjištění udělených oprávnění aplikací).

```
private fun getInstalledPackages(): HashMap<String, String> {
    val PM = packageManager
    val intent = Intent(Intent.ACTION_MAIN, null)
    intent.addCategory(Intent.CATEGORY_LAUNCHER)
    intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or
         Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED
    val infoList = getPackageManager().queryIntentActivities(intent, 0)
    val map = HashMap<String, String>()
    for (resolveInfo in infoList) {
        val actInfo = resolveInfo.activityInfo
        val packageName = actInfo.applicationInfo.packageName
        val label = PM.getApplicationLabel(actInfo.applicationInfo)
        map[packageName] = label
    }
    return map
}

private fun getPermissionsByPackageName(packageName: String):
List<Permissions> {
    try {
        val packageInfo = packageManager.getPackageInfo(packageName,
PackageManager.GET_PERMISSIONS)
        for (i in packageInfo.requestedPermissions.indices) {
            if (packageInfo.requestedPermissionsFlags[i] and
PackageInfo.REQUESTED_PERMISSION_GRANTED != 0) {
                val permissionName = ackageInfo.requestedPermissions[i]
    } } }
    } catch (e: Exception) {
        e.printStackTrace()
    }
    return permissions
}
```

Zdrojový kód 5: Metody pro zjištění udělených oprávnění aplikací

Jak lze z výpisu zdrojového kódu poznat, není příliš složité získat informace o udělených oprávněních aplikaci, a to jakmile známe název balíčku této vybrané aplikace. Pro zjištění názvu balíčku aplikací bylo využito metody `getInstalledPackages()`, která vrací namapované hodnoty názvů balíčků aplikací nainstalovaných v daném zařízení. Metoda byla v průběhu dalšího vývoje aplikace upravena tak, aby vracela pouze takové aplikace, které nepatří mezi systémové aplikace (pomocí kontroly tzv. flags aplikace, pokud je flag nastavena na FLAG_SYSTEM, jedná se o systémovou aplikaci) a aby nevracela aplikaci, která spouští toto mapování nainstalovaných aplikací. Výstupem metody pak tedy byly pouze takové balíčky aplikací, které bylo vhodné testovat na povolené oprávnění.

Obě tyto metody jsou v aplikaci volány v závislosti jedna na druhé. Metoda `getInstalledPackages()` získává nainstalované aplikace okamžitě při spuštění aplikace

8.3.3 Využití asynchronního přístupu

Při získávání dat ze serveru a jejich následnému zobrazování v aktivitě určené pro výsledky testu je využito třídy `AsyncTask`, která přistupuje k datům asynchronně a následně aktualizuje

uživatelské rozhraní (UI - User Interface) aplikace. Jak je totiž známo, není doporučováno provádět dlouhé procesy na UI vlákně, jelikož pak dochází k zamrznutí UI aplikace po tak dlouhou dobu, dokud není daný proces dokončen. V dnešní době již samo IDE Android Studio vypíše chybovou hlášku při nevhodném přístupu k UI vláknu dlouho trvajícími procesy. Třída `AsyncTask` tedy poskytuje možnost provádět dlouhé operace na pozadí aplikace a výsledky operací jsou po dokončení procesu automaticky zveřejněny na UI vlákně bez nutnosti další manipulace s tímto vláknem.

`AsyncTask` je navržen jako pomocná třída, která usnadňuje komunikaci mezi vláknem uživatelského rozhraní a vlákna běžící na pozadí aplikace (tzv. worker vlákno). Toto vlákno běžící na pozadí běžně provádí procesy, které nejsou okamžité, tedy takové, které trvají delší dobu. Nicméně je potřeba si uvědomit, že UI vlákno není možné aktualizovat z kteréhokoliv jiného vlákna. Aby bylo možno upravovat UI vlákno, poskytuje Android několik možností, z nichž jedna je právě s využitím třídy `AsyncTask`. Mezi další možnosti přístupu k UI vláknu se řadí použití metod `Activity.runOnUiThread(Runnable)`, `View.post(Runnable)` nebo `View.postDelayed(Runnable, long)`.

Pro implementaci funkcí třídy `AsyncTask` a možnosti přístupu k UI vláknu, je nutné vytvořit její podtřídu a následně implementovat povinné metody `doInBackground(Params…)` a `onPostExecute(Result)`. Tyto dvě metody nám pak zajišťují správnou funkcionalitu třídy `AsyncTask`. Metoda `doInBackground()` běží v tzv. poolu, kde jsou umístěny i další vlákna, běžící na pozadí aplikace. Tato metoda zajišťuje základní přístup k serveru a k jeho přístupovým bodům. Abychom mohli aktualizovat UI podle dat, která získáme z databáze, je nutné použít metodu `onPostExecute()`, která posílá výsledky získané z metody `doInBackground()` do vlákna, které se stará o UI. Díky této metodě je umožněna bezpečná aktualizace UI ve spuštěné aktivitě. Pro spuštění celého asynchronního přístupu k serveru je nutné pouze volat metodu `execute()` z běžícího UI vlákna aplikace.

Při využívání podtřídy `AsyncTasku`, je možno implementovat další metody, jako jsou `onPreExecute()` a `onProgressUpdate()`. Metoda `onPreExecute()` je volána před tím, než je spuštěno samotné vlákno na pozadí. Nejčastěji je této metody využito pro nastavení úlohy, která bude běžet na pozadí (například zobrazení ukazatele průběhu). Metoda `onProgressUpdate()` je volána, pokud je využito metody `publishProgress()` v metodě `doInBackground()` a je využívána pro zjištění postupu vlákna a zobrazení tohoto stavu v UI vlákně. [29]

8.3.4 Odebrání udělených oprávnění aplikace

Do aplikace bylo zvažováno zabudování možnosti odebrání již dříve udělených oprávnění. Po delším hledání možností pro odebrání udělených oprávnění však došlo ke zjištění, že odebrat oprávnění jiným způsobem než pomocí nastavení není zatím možné. Jediné, čeho se dá využít, avšak pouze pro testovací účely, je níže zobrazený shell skript (viz Zdrojový kód 6: Skript pro odebrání oprávnění aplikace).

```
$ adb shell pm revoke <package_name> <permission_name>
```

Zdrojový kód 6: Skript pro odebrání oprávnění aplikace [29]

Pro využití tohoto skriptu je nutné na zařízení provést několik kroků, z nichž jeden je připojení zařízení k počítači, kde jsou provedeny následující kroky pro správné fungování vývojového nástroje ADB (Android Debug Bridge) [31]. Prvním krokem pro správné fungování nástroje ADB je nutnost umožnění přístupu do sekce pro vývojáře v nastavení zařízení s OS Android a následné povolení položky ladění USB. Následujícím krokem je stažení Android SDK Platform Tools a v jejichž složce otevřít příkazový řádek, do kterého již lze psát nejrůznější příkazy nástroje ADB. [32]

Z důvodu, jakým stylem je využíváno nástroje ADB, lze využít tohoto nástroje pouze v zařízení připojenému k počítači a není tedy možné provádět například zmíněné odebírání oprávnění v rámci aplikace programově.

Tento nástroj však alespoň usnadňuje komunikaci mezi zařízením se systémem Android a osobním počítačem a poskytuje mnoho funkcí, které lze v konzoli počítače (ze složky platform-tools) vypsat příkazem `adb --help`.

8.3.5 Odebrání nainstalované aplikace

Uživateli je jako jedna z hlavních možností po zjištění skóre aplikace nabídnuta možnost odinstalování vybrané aplikace ze zařízení. Metoda zobrazena v následujícím zdrojovém kódu (viz Zdrojový kód 7: Metoda pro odinstalování aplikace) provede odinstalování aplikace na základě uvedeného jména balíčku aplikace, který je dle zásad vývoje aplikací unikátní v celém systému. Nemůže tedy omylem dojít ke smazání jiné aplikace.

```
private fun uninstallApplication(packageName: String){
    val intent = Intent(Intent.ACTION_UNINSTALL_PACKAGE)
    intent.data = Uri.parse("package:$packageName")
    intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK
    this@TestResults.startActivity(intent)
}
```

Zdrojový kód 7: Metoda pro odinstalování aplikace

Po použití této funkce však bylo nutné uživatele (po dokončení samotné odinstalace vybrané aplikace) odnavigovat zpět do hlavní aktivity aplikace. Pro tuto potřebu bylo nutné vytvořit tzv. broadcast receiver, který umí zachytávat různé akce. Pro naše potřeby bylo dostačující rozeznání akce týkající se odstranění balíčku jakékoliv aplikace, která je zobrazena na následujícím zdrojovém kódu (viz Zdrojový kód 8: Využití broadcast receiveru). Filtrovat však lze i mnoho dalších akcí OS Android, jako je například přidání nového balíčku nebo připojení zařízení ke zdroji napájení.

```
val receiver = object : BroadcastReceiver() {
   override fun onReceive(c: Context, i: Intent) {
      val intent = Intent(this@TestResults, MainActivity::class.java)
      startActivity(intent)
   }
}
val intentFilter = IntentFilter()
intentFilter.addAction(Intent.ACTION_PACKAGE_REMOVED)
intentFilter.addDataScheme("package")
registerReceiver(receiver, intentFilter)
```

Zdrojový kód 8: Využití broadcast receiveru

9 Testování aplikace

Pro testování správného ohodnocení testovaných aplikací bylo nejprve nutné nalézt takové aplikace, které byly označeny jako škodlivé. Oficiální informace vydávané společností Google bohužel nebylo možné najít, v průběhu hledání však byla nalezena webová stránka mapující a testující Android aplikace [27]. Z této webové stránky je možné stahovat instalační APK (Android Package File) soubory (soubor s koncovkou .apk) vybraných aplikací a ty následně nainstalovat do vybraného mobilního zařízení. Jelikož se jednalo o škodlivé aplikace ověřené několika vývojáři či testery, byly tyto aplikace zařazeny do tzv. učící množiny aplikací. Do této množiny aplikací bylo následně přidáno několik dalších aplikací, které byly určené jako neškodné.

Poměr škodlivých a neškodlivých aplikací byl ve výsledné množině přesně 1:1. Přesné počty udávají, že došlo k instalaci dvaceti škodlivých aplikací, ke kterým bylo následně přidáno dvacet neškodných aplikací. Výběr škodlivých aplikací byl náročný, jelikož v mnoha případech docházelo při stahování k detekci škodlivého kódu antivirovým programem, a nedošlo tak ke stažení instalačního souboru. A jak bylo následně zjištěno, ani v případě, že se podařilo instalační soubor aplikace stáhnout, neznamenalo to, že ji bude možné do emulovaného zařízení nainstalovat. Přibližně 50% stažených instalačních souborů bylo nějakým způsobem poničeno nebo jim chyběly certifikáty a instalace aplikace tak nemohla být dokončena. Z přibližně šedesáti vytipovaných škodlivých aplikací tak došlo ke stažení a skutečnému nainstalování výše zmíněných dvaceti aplikací.

9.1 Instalace APK souborů do emulátoru

Ze všeho nejdříve bylo pro provedení testování ohodnocení aplikací nutné přidat do emulátoru zařízení vybrané škodlivé aplikace, které budou tvořit výše zmíněnou učící množinu. Toto přidání APK souborů škodlivých aplikací bylo provedeno pomocí programu Windows PowerShell, který byl spuštěn ve složce platform-tools umístěné v adresáři Android SDK (přesněji C:/Users/Erika/AppData/Local/Android/Sdk/platform-tools). Pro jednoduchost navigace k APK souborům aplikací, které chceme na zařízení nainstalovat, je možné je do této složky vložit.

Pro nainstalování vybraných APK souborů byl v otevřeném PowerShellu zadán příkaz, který je zobrazen na následujícím zdrojovém kódu (viz Zdrojový kód 9: Instalace APK souborů do emulátoru). Parametr `-g` v tomto příkazu určuje povolení všech oprávnění, které daná aplikace požaduje.

```
./adb install -g <název_souboru.apk>
```

Zdrojový kód 9: Instalace APK souborů do emulátoru

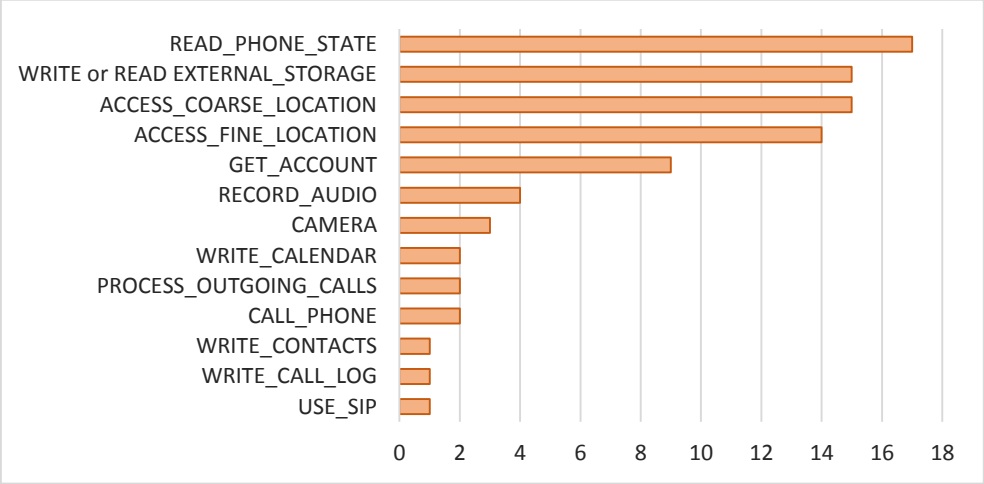
Po úspěšném nainstalování docházelo k výpisu zprávy „success“, tedy úspěch. V případě neúspěšné instalace pak docházelo k vypsání přesného důvodu, proč se instalace nezdařila. Nejčastěji docházelo ke zprávě, která informovala, že aplikace neobsahuje nativní knihovnu pro

danou architekturu procesoru. S méně častým výskytem se pak zobrazovala informace o neplatném certifikátu aplikace.

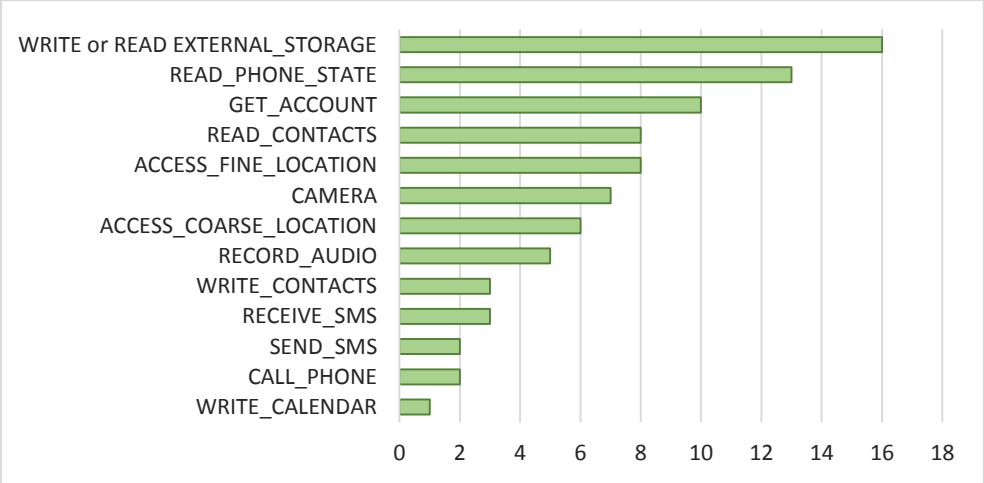
9.2 Využití nebezpečných oprávnění aplikacemi

Následující dva obrázky (viz Obrázek 14: Počty jednotlivých oprávnění využívaných ve škodlivých aplikacích a Obrázek 15: Počty jednotlivých oprávnění využívaných v neškodných aplikacích) zobrazují porovnání počtů jednotlivých udělených nebezpečných oprávnění ve škodlivých a neškodných aplikacích. Obrázky zobrazují porovnání počtů pouze pro prvních 13 příček, jelikož zbývající příčky neobsahují vysoké počty u jednotlivých oprávnění.

Při zaměření se na jednotlivé nebezpečné oprávnění, která jsou udělována jak ve škodlivých tak v neškodných aplikacích, je možné si povšimnout, že počty těchto oprávnění jsou v některých případech velmi odlišné. Podíváme-li se na prvních pět pozic v každém z obrázků, zjistíme sice, že první dvě příčky jsou obsazené stejnými oprávněními (jen v opačném pořadí). Na dalších příčkách se však u nebezpečných aplikací objevují jiná oprávnění než u neškodných aplikací.



Obrázek 14: Počty jednotlivých oprávnění využívaných ve škodlivých aplikacích



Obrázek 15: Počty jednotlivých oprávnění využívaných v neškodných aplikacích

Jak již bylo zmíněno dříve, všechna tato oprávnění nemusí nutně být relevantní pro detekci nebezpečné aplikace a tím mohou mít za následek nesprávné klasifikační výsledky. Takovéto výsledky mohou mít zaváděcí účinek pro algoritmus učení. Převážně z tohoto důvodu byly do výpočtu klasifikace zahrnuty relevance jednotlivých nebezpečných oprávnění.

9.3 Výsledky klasifikace aplikací

V následující tabulce (viz Tabulka 4: Výsledky ohodnocení aplikací) jsou přehledně zobrazeny výsledky klasifikací u nově testovaných aplikací. Je zde vidět, že vybraný klasifikační algoritmus kNN má své nedostatky, respektive není možné se stoprocentní úspěšností určit škodlivost vybrané aplikace pouze na základě udělených oprávnění.

Tabulka 4: Výsledky ohodnocení aplikací

	Počet aplikací	**Správně ohodnocené aplikace (počet)**	**Nesprávně ohodnocené aplikace (počet)**	**Správně ohodnocené aplikace (%)**	**Nesprávně ohodnocené aplikace (%)**
○ celkem	20	15	5	75%	25%
× škodlivé	10	9	1	90%	10%
✓ neškodné	10	6	4	60%	40%

Tabulka udává, že procento správně ohodnocených aplikací se pohybuje okolo hodnoty 75%, přičemž u škodlivých aplikací byla vyšší úspěšnost správné klasifikace než u aplikací neškodných. Tento výsledek však mohl být ovlivněn tím, že učící množina se v základu skládala pouze ze 40 prvků (20 škodlivých a 20 neškodných aplikací). V průběhu klasifikace dalších aplikací se tato učící množina zvětšovala vždy o nově ohodnocené aplikace. Z tohoto důvodu se může stát, že algoritmus začne průběžně konvergovat k nesprávným, falešně pozitivním či falešně negativním, výsledkům.

9.4 Původ nebezpečných aplikací

Aplikace, které byly dle zdroje [27] označeny za škodlivé, často obsahovaly pornografický materiál. Mezi škodlivými aplikacemi byly nelezeny i takové aplikace, které měly stejný název jako jiné známé a neškodlivé aplikace (facebook apod.). Tyto stejnojmenné aplikace však obsahovaly škodlivé funkce, které originální aplikace neobsahují.

Velkou mírou byly škodlivé aplikace zastoupeny pod názvy, které byly psány jinou abecedou (azbukou či čínskými znaky). Dá se tak předpokládat, že velká část škodlivých aplikací pochází právě ze zemí, které používají tyto abecedy. Jelikož však nelze reálně zjistit, odkud aplikace pocházejí, tak se jedná pouze o předpoklad a nemusí to nutně být pravda.

10 Závěr

Cílem diplomové práce bylo teoreticky popsat problematiku bezpečnosti a monitorování zařízení s operačním systémem Android a následně vytvořit funkční aplikaci pro možnost dohledu nad mobilním zařízením. Pro seznámení s touto problematikou byla vyhrazena kapitola číslo 2, která se zabývá jak současným stavem existujících aplikací zabývajících se podobným tématem, tak seznámením čtenáře s existujícími nebezpečnými oprávněními a s jejich riziky. Tato nebezpečná oprávnění totiž mohou potenciálně ovlivnit soukromí uživatele nebo normální provoz zařízení, čehož si uživatel aplikace nemusí být vědom, ač s jejichž povolením musí výslovně souhlasit.

Teoretická část práce je popsána v kapitolách 3 až 6. Jsou zde popsány základní vlastnosti OS Android, jako je jeho architektura a správa napájení, jaké obsahuje bezpečnostní prvky a služby poskytované společností Google, která zastřešuje OS Android. Dále se práce věnuje zabezpečení OS Android z hlediska platformy, systému a jádra, včetně popisu aplikačního sandboxu. Následně je zde uveden přehled existujících a využívaných oprávnění Android aplikací, a to jak z hlediska existujících skupin oprávnění, tak i z hlediska postupu schválení oprávnění. Závěrem teoretické části práce jsou popsány komponenty Android aplikace, které se dělí na aktivity, služby, poskytovatele obsahu, přijímače vysílání a záměry. Do této kapitoly je zařazen i popis souboru AndroidManifest.xml, kde dochází ke specifikaci všech těchto komponent a jejich využití v každé implementované aplikaci.

Praktickou částí diplomové práce se zabývají kapitoly 7 až 9, kde je popsán prvotní návrh aplikace, obsahující diagramy jako je relační datový model databáze, usecase diagram pro popis klíčových funkcí klientské aplikace, nebo třídní diagram pro popis objektů využívaných jak na straně klienta, tak na straně serveru. V této kapitole je zobrazeno i základní GUI klientské aplikace. Implementaci projektu je věnována kapitola 8, kde je popsán vývoj klíčových prvků všech částí aplikace. Mezi tyto prvky je zařazeno monitorování povolených oprávnění a následný výpočet skóre dané aplikace na základě povolených oprávnění aplikace. Poslední kapitola zabývající se praktickou částí je věnována testování výsledné aplikace, včetně přehledného zobrazení výsledků v grafech a tabulkách.

Cíle stanovené v úvodu práce byly úspěšně dosaženy. V průběhu implementace aplikace bylo nalezeno několik málo možností vylepšení, a to jak vzhledu a orientace v aplikaci, tak přidání dalších funkcí, které původně nebyly v zadání specifikovány. Jedná se tak například o možnost odebrání již uděleného oprávnění aplikace, při kterém došlo ke zjištění, že OS Android zatím nepodporuje programové odebírání udělených oprávnění.

Mezi možné vylepšení aplikace a pokračování v tomto tématu může být zařazeno rozšíření aplikace o možnost otestování ještě nenainstalovaných aplikací, a to s pomocí Google Play API.

Reference

[1] DEVELOPERS. *Application permissions*. [online]. [cit. 2018-11-14]. Dostupné z: <https://developer.android.com/guide/topics/permissions>

[2] TANG, X., WU, D., LIN, Y., GAO, D. *Towards Dynamically Monitoring Android Applications on Non-rooted Devices in the Wild*. [online]. [cit. 2018-11-14] Dostupné z: <https://dl.acm.org/citation.cfm?doid=3212480.3212504>

[3] KELLEY PG., CONSOLVO S., CRANOR L., JUNG J., SADEH N., WETHERALL D., *A conundrum of permissions: Installing applications on an Android smartphone.* [online]. [cit. 2018-11-15]. Dostupné z: <https://link.springer.com/chapter/10.1007/978-3-642-34638-5_6>

[4] CHESTER P., KRUTZ D., PEREZ C., JONES CH. *M-Perm*. [online]. [cit. 2018-11-15]. Dostupné z: <https://mperm.github.io/index.html>

[5] YAN LK., YIN H. *DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis*. [online]. [cit. 2018-11-15]. Dostupné z: <https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/yan>

[6] LANTZ P., DESNOS A., YANG K. *DroidBox: Android application sandbox*. [online]. [cit. 2018-11-15]. Dostupné z: <https://github.com/pjlantz/droidbox>

[7] STATISTA. *Global mobile OS market share*. [online]. [cit. 2018-10-30]. Dostupné z: <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>

[8] DEVELOPERS. *Platform Architecture*. [online]. [cit. 2018-10-30]. Dostupné z: <https://developer.android.com/guide/platform>

[9] ANDROID OPEN SOURCE PROJECT. *Power Management.* [online]. [cit. 2019-01-02]. Dostupné z: <https://wladimir-tm4pda.github.io/porting/power_management.html>

[10] SOURCE. *Power Management.* [online]. [cit. 2019-01-02]. Dostupné z: <https:/source.android.com/devices/tech/power/mgmt>

[11] SOURCE. *Security*. [online]. [cit. 2018-12-11]. Dostupné z: <https://source.android.com/security>

[12] SOURCE. *Application Sandbox*. [online]. [cit. 2018-11-12]. Dostupné z: <https://source.android.com/security/app-sandbox>

[13] SOURCE. *System and kernel Security*. [online]. [cit. 2018-11-12]. Dostupné z: <https://source.android.com/security/overview/kernel-security>

[14] WENLIANG DU. *Android Device Rooting Lab*. [online]. [cit. 2019-04-20]. Dostupné z: <http://www.cis.syr.edu/~wedu/seed/Labs_Android5.1/Android_Rooting/Android_Rooting.pdf>

[15] SOURCE. *Encryption*. [online]. [cit. 2018-12-10]. Dostupné z: <https://source.android.com/security/encryption>

[16] DEVELOPERS. *Device administration overview*. [online]. [cit. 2018-12-15]. Dostupné z: <https://developer.android.com/guide/topics/admin/device-admin>

[17] DEVELOPERS. *Permissions overview.* [online]. [cit. 2019-03-09]. Dostupné z: <https://developer.android.com/guide/topics/permissions/overview#permissions>

[18] CLINTON TEEGARDEN. *Runtime Permissions: Best Practices and How to Gracefully Handle Permission Removal.* [online]. [cit. 2019-03-18]. Dostupné z: <https://www.captechconsulting.com/blogs/runtime-permissions-best-practices-and-how-to-gracefully-handle-permission-removal>

[19] DEVELOPERS. *Application fundamentals*. [online]. [cit. 2018-11-07]. Dostupné z: <https://developer.android.com/guide/components/fundamentals>

[20] MEDIUM. *Activity lifecycle in Android applications*. [online]. [cit. 2018-11-10]. Dostupné z: <https://medium.com/sketchware/activity-lifecycle-in-android-applications-1b48a7bb584c>

[21] HORDĚJČUK V. *JPA (Java Persistence API).* [online]. [cit. 2019-03-23]. Dostupné z: <http://voho.eu/wiki/java-jpa/>

[22] HIBERNATE. *Hibernate ORM, Your relational data, objectively*. [online]. [cit. 2019-03-23]. Dostupné z: <http://hibernate.org/orm/>

[23] POSTGRESQL. *About PostgreSQL*. [online]. [cit. 2019-03-23]. Dostupné z: <https://www.postgresql.org/about/>

[24] PGADMIN. *What is pgAdmin*. [online]. [cit. 2019-03-23]. Dostupné z: <https://www.pgadmin.org/faq/#1>

[25] S.M.H. AL-TMEEMY ET AL. *Data analysis.* [online]. [cit. 2019-04-03]. Dostupné z: <http://repository.um.edu.my/37709/1/1-s2.0-S0263786311001542-main.pdf> str. 6

[26] BRONSHTEIN A. *A Quick Introduction to K-Nearest Neighbors Algorithm*. [online]. [cit. 2019-04-03]. Dostupné z: <https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7>

[27] KOODOUS. [online]. [cit. 2019-02-26]. Dostupné z: <https://koodous.com/>

[28] DEVELOPERS. *Android 6.0 Changes.* [online]. [cit. 2019-03-03]. Dostupné z: <https://developer.android.com/about/versions/marshmallow/android-6.0-changes>

[29] DEVELOPERS. *AsyncTask.* [online]. [cit. 2019-04-20]. Dostupné z: <https://developer.android.com/reference/android/os/AsyncTask>

[30] ANDROID BLOG. *Marshmallow – Grant and Revoke permissions with adb*. [online]. [cit. 2019-03-19]. Dostupné z: <https://srikanthkommineni.com/2015/10/14/marshmallow-grant-and-revoke-permissions-with-adb/>

[31] LINEAGEOS WIKI. *Using ADB and fastboot.* [online]. [cit. 2019-03-19]. Dostupné z: <https://wiki.lineageos.org/adb_fastboot_guide.html>

[32] XDA DEVELOPERS. *How to quickly install and use ADB*. [online]. [cit. 2019-03-20]. Dostupné z: <https://www.xda-developers.com/quickly-install-adb/>